UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

19|20

Trabajo fin de grado

Desarrollo de sistema de análisis automático de phishing

Sergio Frontera Díaz de Quintana

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Desarrollo de sistema de análisis automático de phishing

Autor: Sergio Frontera Díaz de Quintana
Tutor: Jaime López Sánchez
Ponente: Fernando Díez Rubio

julio 2020

Sergio Frontera Díaz de Quintana
Desarrollo de sistema de análisis automático de phishing

Sergio Frontera Díaz de Quintana
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mis padres y a mi hermana

«Las empresas invierten millones en firewalls, cifrado y dispositivos para acceder de forma segura, y es dinero malgastado, porque ninguna de estas medidas corrige el eslabón más débil de la cadena.»

Kevin Mitnick

Agradecimientos

Tras varios meses de trabajo y esfuerzo, me gustaría agradecer el apoyo y la ayuda ofrecida por Jaime López, la persona que ha planteado esta idea del trabajo sobre la automatización del phishing y la que se ha encargado de acompañarme durante todo el proyecto para que resultase satisfactorio. He aprendido mucho de él, tanto en el ámbito más técnico de la informática como en sus valores de trabajo y humanos. Siempre ha sacado tiempo para dedicarme, y nunca ha puesto ningún tipo de pega cuando le he solicitado una tutoría o reunión de cualquier tema relacionado con el TFG. Por causas de la pandemia que está golpeando al planeta entero, apenas estuve un mes con él de forma presencial desarrollando el trabajo en el equipo de fraude de la empresa, aunque las herramientas telemáticas nos han facilitado la comunicación entre ambos. Además, hacer el TFG con esta empresa me ha abierto la oportunidad de continuar en ella en mi nueva etapa laboral.

Asimismo, agradezco a Fernando Díez, ponente de este trabajo, encargado de mediar entre la empresa en la que he hecho el TFG y la Universidad, de prestarse desinteresadamente por si necesitaba cualquier tipo de ayuda o consejo, comprometiéndose durante todo el periodo que ha durado el desarrollo del trabajo. Hemos intercambiado numerosos mensajes con los que he conseguido afinar y orientar mejor la memoria del TFG.

Finalmente, debo agradecer a mi padre la ayuda que me ha brindado supervisando algunos aspectos de estilo y redacción. Aún me queda mucho por mejorar.

Resumen

La tecnología avanza rápido y la evolución de Internet sigue en pleno apogeo. La red brinda cientos de oportunidades a los ciberdelincuentes para que, a través de diferentes técnicas, cometan actividades ilícitas con las que robar nuestra información más sensible como datos bancarios, credenciales, u otros datos personales. La ciberseguridad es un sector dentro de la informática que tiene cada vez más relevancia y debe estar constantemente alerta para poder combatir los cientos de fraudes online que se propagan diariamente por la red. El fraude digital más usual es el phishing, el cual consiste en suplantar la identidad de una persona o una empresa, engañando a los usuarios para sustraer sus datos.

Hoy en día, se aplican distintas técnicas para combatir el phishing. Dado que el medio más utilizado para propagarlo es el correo electrónico que utiliza el protocolo SMTP, que presenta deficiencias de seguridad, se emplea el protocolo DMARC como complemento, que integra los protocolos SPF y DKIM, securizando la comunicación, reduciendo así el impacto del phishing. Adicionalmente, también ayuda a identificar numerosos casos de phishing la utilización de la cabecera “referer” del protocolo HTTP, la cual permite detectar redireccionamientos de páginas web fraudulentas a los dominios legítimos.

En este trabajo se ha planteado una solución basada en una arquitectura software con capacidad de ser implementada en cualquier equipo o máquina, compuesta de dos plataformas (un frontal web y una API) con la que poder interactuar con ella. El fin principal se basa en analizar páginas web a partir de sus recursos, a los que se les aplica un tipo de función hash para obtener un identificador único de cada uno de ellos con el propósito de compararlos con los almacenados en la base de datos.

Para almacenar información en la base de datos, el software permite agregar información sobre los recursos de una página web, tanto legítima como no legítima, que ayude a posteriori en los análisis de las URL sospechosas, pudiéndose visualizar también la información de una compañía contenida en la base de datos. Dispone además, de un algoritmo de “matches” que hace referencia a la relevancia o importancia de un recurso web sobre los demás.

Palabras clave

Phishing, kits de phishing, fraude online, automatización, ciberseguridad, IoCs, hashes, Docker, Flask, Python, software, correo electrónico, protocolo DMARC

Abstract

Technology is advancing ever more rapidly and Internet is still in full swing and social media network provides hundreds of opportunities for cybercriminals, so that using different techniques, illegally obtain personal identity and/or financial information, among other sensitive information. Cybersecurity is a valuable field of study in the IT industry which is getting more and more importance due to the necessity of protecting any organization against multiple online frauds that spread all over Internet. One of the most frequent and easiest way to commit those frauds is through Phishing techniques, which consist in sending an apparently legitimate message to an individual or a company to steal their data.

Nowadays, are designed different techniques and methodologies in order to fight this kind of fraud. Given that mails are the most frequent ways of spreading fraud and deception that use a weak security protocol SMTP, the DMARC protocol is complementary added to reinforce it. DMARC, which integrates SPF and DKIM protocols, secures the communication among sender and receiver, reducing the risk of Phishing. Additionally, to the above, the use of “referer” header of the HTTP protocol also helps to avoid many cases of phishing, which allows detection of redirects from fake web pages to legitimate domains.

This academic study contemplates a solution based on a software architecture that has the possibility of being implemented on any type of computer. It is made up of two platforms (frontal web and an API) to facilitate interaction. The main end of this solution is to analyze a web page from its particular resources, to which a type of “hash” function is applied to obtain a unique identifier for each of them with the purpose of comparing them with those stored in the database.

In order to store information in the database, the software developed allows the user the addition of information about those resources presented in a web page, both legitimate and non-legitimate, that helps later on to analyze suspicious URLs. This software also enables us to consult specific company data contained in the database. It also counts with a “matches” algorithm that refers the importance of one specific web resource to others.

Keywords

Phishing, phishing kits, online scams, automation, cybersecurity, IoCs, hashes, Docker, Flask, Python, software, email, DMARC protocol

Índice

Listas

Lista de códigos

Lista de figuras

1

Introducción

En la actualidad, debido al avance de las tecnologías y la transformación digital que está experimentando la sociedad, las interacciones entre las personas y el mundo digital son cada vez más amplias.

El **uso masivo de Internet**, como medio de comunicación por el que se mueve un ingente volumen de datos e información, permite acercarse a cualquier punto de la geografía mundial a través de múltiples puntos de acceso e interconexión. Por la red circula **información altamente sensible** como datos personales, económicos, técnicos, estratégicos, de seguridad, etc., de los que nuestro mundo globalizado se ha hecho altamente dependiente.

Como consecuencia de este progreso digital, donde la información fluye ampliamente por las redes, la carrera para acceder a ella, buscando vulnerabilidades en los sistemas hace que los riesgos de comisión de **fraudes y delitos informáticos** sean cada vez más frecuentes. La **ciberdelincuencia** ha ido aumentando durante estos años de progreso, acompasando su actividad a ese crecimiento, aprovechando el incremento de usuarios que se conectan a la red, a resultas de esa globalización.

Estos delincuentes expertos de la era digital, que disponen de amplios conocimientos en materia de seguridad y comunicaciones, buscan sin descanso cualquier resquicio o falla en la seguridad que les permita acceder al contenido de las comunicaciones, ordenadores, tablets, móviles, servidores o cualquier otro dispositivo conectado a la red con el que puedan interactuar digitalmente. Los ciberdelincuentes son cada vez más eficientes, actúan bajo identidades anónimas en Internet, evitando dejar huellas digitales de sus pasos y emplean equipos informáticos muy sofisticados y potentes para alcanzar sus objetivos.

Ante las incesantes amenazas, riesgo de estafas, daños en instalaciones particulares y estratégicas, a la imagen de las personas, a la venta y divulgación ilícita de información confidencial, se precisa una lucha constante para hacer frente a las cada vez más ingeniosas formas de fraude. Por ello, la ciberseguridad debe reforzar permanentemente su capacidad de respuesta, adaptándose rápidamente a las nuevas formas de ataque de este tipo de actividades ilegales en el espacio digital.

Los objetivos de los ciberdelincuentes son muy variados, utilizan el robo de información personal

para **suplantar la identidad** de la víctima o para **venderla en el mercado negro** a terceros, almacenándose todos ellos en grandes bases de datos con el fin de enviarles propaganda o publicidad masiva de cualquier índole. Otros, buscan **bloquear el acceso a los sistemas de archivos** de una empresa cifrando el contenido de los ordenadores, **para pedir posteriormente un rescate**, ataque conocido como ransomware. En estos casos la recomendacion que se da es no pagar el rescate. Sin embargo, muchas pymes, las principales víctimas de esta actividad, debido al estrés e intranquilidad que produce este tipo de situaciones acaban desistiendo y pagan el dinero impuesto (pocas veces negociable) por el atacante. A veces no consiguen ni siquiera recuperar los datos después de pagar, y, además, provoca que los delincuentes se crezcan y reiteren el mismo delito en otra empresa del mismo perfil. Otros fines que tienen los ciberdelincuentes es recibir **contenido de carácter sexual** de sus víctimas para lucrarse después distribuyéndolo por el Internet profundo (*Deep Web*). Para ello la extorsionan a través de redes sociales o mediante un intercambio de correos electrónicos.

Pueden recurrir a una amplia variedad de recursos o artimañas para cometer delitos como el envío de un **ejecutable malicioso**, un ***keylogger*** para robar credenciales y contraseñas, etc., pero para que los usuarios hagan click en estos ejecutables o *malwares*, debes de llamar su atención.

Desde luego que es más sencillo obtener una contraseña directamente dada por la propia víctima, que robar una base de datos y hacer un ataque por fuerza bruta para obtener la clave de un usuario. Por ello, el método más efectivo para conseguir estos objetivos es utilizar la **ingeniería social**, donde se busca el fallo humano. Esta forma de engaño mantiene la premisa de que manipular a las personas es más fácil que manipular a una máquina.

Una de las formas de ingeniería social que se va a estudiar en este documento es **el phishing**, el fraude digital más extendido entre los ciberdelincuentes. Los phishers –los usuarios que están detrás del ataque–, **se hacen pasar por una persona o entidad** de quien la víctima no tendría motivos para desconfiar, como por ejemplo un miembro de su empresa, una entidad bancaria, o cualquier otra organización conocida por la víctima y a través de un correo electrónico, por ejemplo, le piden que inicie sesión o introduzca datos sensibles en una página falsificada suplantando otra legítima **para robar credenciales o números de tarjetas de crédito**.

Por ello es tan importante encontrar una tecnología o unas herramientas que estén a la altura de este problema, y que ayuden en lo máximo posible a paliar estos ataques. Disponer de estos software sofisticados para la detección del phishing es de vital importancia. No es para menos, pues según un artículo de *CSO United States*, portal especializado en seguridad, se crea un nuevo phishing cada 20 segundos [1]. Además, en torno al 91 % de las violaciones de seguridad como ***ransomware***, o robos de datos se han cometido a través de un phishing [2].

Hay **dos formas de detectarlos**: por **vía activa**, en la que se instalan mecanismos de detección en la infraestructura de entidades, empresas o particulares que pueda ser una víctima potencial de phishing; o por **vía pasiva** donde se hace una búsqueda en fuentes abiertas, sea en redes sociales,

foros de phishing, fuentes online que suministran URLs potencialmente maliciosas de posibles casos de phishing que ha reportado la comunidad.

En este Trabajo de Fin de Grado se va a trabajar con la segunda forma de detección de phishing, la vía pasiva. Para ello, es necesario la automatización de ciertos procesos, para facilitar la detección y el análisis de las búsquedas realizadas, por lo que el software requerido se basa en la automatización procesos de detección y análisis de URLs potencialmente maliciosas.

1.1. Motivación y objetivos

La motivación de este trabajo viene dada fundamentalmente por la **necesidad de automatizar el proceso de detección de URLs altamente sospechosas y análisis de phishing**, buscando una forma eficaz y escalable, la cual sirva como punto de partida para que en un futuro puedan implementarse algoritmos y recursos más avanzados. Aunque existen herramientas automatizadas pertenecientes a grandes empresas, en la mayoría de los casos se realizan análisis manuales de estas URLs lo que ralentiza el proceso de eliminación de los phishing, y, por tanto, cuanto más tiempo pase, mayor exposición al fraude y más perdidas se producen en las compañías afectadas o por parte de sus clientes. En consecuencia, la finalidad más práctica que tiene este trabajo a futuro es **tratar de minimizar el tiempo de vida de un phishing** desde que se propaga hasta que se elimina finalmente.

Objetivos

A continuación, se enumeran los tres objetivos que se quieren abordar en este proyecto. Están enumerados según la prioridad que se les ha otorgado a cada uno.

O-1.– Crear una **arquitectura base con capacidad de ser escalada**, que reciba muchas URLs por minuto, identificando cuáles son phishing y cuáles no y a qué empresa suplantan (el envío de las URLs al sistema es ajeno al trabajo, no es propósito de este software).

O-2.– Almacenar información de estas URLs para buscar relaciones entre los distintos kits de phishing para ayudar a posteriori en la identificación de la empresa a la que suplantan. (*Un kit de phishing* proporciona a los ciberdelincuentes las herramientas necesarias para crear páginas web falsas muy convincentes y lanzar campañas maliciosas a gran escala vía correo electrónico o por redes sociales).

O-3.– Debido a que el análisis y el almacenamiento de esta información puede ser lenta, **el tercer objetivo consiste en que la ejecución de estos análisis sea eficiente**, para ello, se plantean dos soluciones: una a nivel de proceso donde se utilizarán hilos para reducir el tiempo de resolución de los análisis; y la segunda será crear un sistema distribuido de nodos y colas que permitirá una escalabilidad horizontal.

1.2. Estructura del documento

En los próximos apartados se estudiará con más detalle el fenómeno del phishing: qué es, en qué consiste y el origen del término. Después, se explicará su contexto actual, qué son los kits de phishing, los tipos de phishing y perfiles de las víctimas. Asimismo, se nombrarán las técnicas y metodologías de los phishers para acometer sus objetivos. Se estudiará la evolución del phishing hasta la actualidad, reservando un apartado de lo ocurrido este año 2020 y cómo se aprovechan los estafadores de la pandemia del coronavirus para inducir temor e influir en las personas.

Se dedica una sección a la parte correspondiente a la ciberseguridad, qué técnicas se aplican para identificar o detectar estos phishing, qué fuentes existen para localizar phishing o verificar si una página es phishing.

Posteriormente, se expondrá todo lo relevante a la solución que se ha propuesto para este TFG. Se explicará cuál es el elemento clave que nos va ayudar a hacer el análisis, cómo está modularizado y organizado el software, qué componentes tiene, qué función tiene cada uno y qué herramientas se utilizan para conseguirlo. En este mismo capítulo donde se explican todos los detalles funcionales de la arquitectura propuesta, se incluye además, en una nueva sección, las pruebas y resultados.

Finalmente, se citarán las conclusiones y el trabajo a futuro en la misma línea de investigación.

2

El Phishing

El phishing hace referencia a uno de los métodos de fraude online más extendido hoy en día. Consiste en un conjunto de técnicas que llevan al delincuente a engañar y manipular a la víctima, suplantando la identidad de una persona o una empresa, dando la apariencia de proceder de una fuente de confianza. Estas técnicas persiguen capturar información personal tales como credenciales o datos de cuentas y tarjetas bancarias para sustraer dinero. Otras formas de hacerse con el dinero de la víctima es engañando de manera que acabe instalando un troyano, un *malware* o un *ransomware*, para pedir posteriormente un rescate. Todos ellos actúan bajo un procedimiento basado en la ingeniería social.

La ingeniería social es una práctica que consiste en la manipulación de las personas a partir de técnicas psicológicas o habilidades sociales sustentándose en la propia naturaleza humana. En el caso del phishing, contactan con las víctimas a través del correo electrónico redactando mensajes ingeniosos requiriendo una actuación urgente solicitando cierta información sensible a la víctima.

Por ejemplo, ponen la excusa de que ha surgido un fallo de seguridad en la web, y necesitan los datos cuanto antes. También pueden adjuntarles archivos maliciosos con un título y un mensaje llamativo para que se los descarguen. Hoy en día, los phishing no solo proceden vía email, sino que, emplean otros tipos de sitios online como las redes sociales, salas de chat o mensajería instantánea, etc.

La figura 2.1 es un ejemplo genérico de correo de phishing de una institución bancaria, en este caso del Banco Santander. En este caso, se manda un correo al cliente avisándole que ha habido una falla de seguridad con la conexión de su última sesión en la aplicación, y que por seguridad, para solucionar el percance técnico, le solicitan asegurarse de cerrar dicha sesión. Para hacerlo creíble, muestran una imagen del logotipo del banco, así como que avisan de que el correo es unidireccional, es decir, no permite contestaciones.

El origen etimológico del phishing proviene del inglés *fishing*, que significa “pesca” y hace alusión al concepto que se usa en el gremio de “picar el anzuelo”.

Figura 2.1: Ejemplo de correo de phishing

2.1. Estado del arte

2.1.1. Kits de phishing

Para entender bien el contexto actual del phishing, es necesario conocer el *modus operandi* de los estafadores, que se esconden en la red para cometer sus delitos. En el caso del phishing, el phisher es la persona que está detrás de un ataque. La mayoría de los phishers son expertos en la materia, sin embargo, la tendencia es que aparezcan más usuarios con menor conocimiento técnico gracias a las facilidades que ofrecen los kits de phishing, cada vez más avanzados y más fáciles de usar. Esto se debe a que cada vez existe más competencia en el sector clandestino de Internet y para llegar a más personas se ven obligados a dar más facilidades a los clientes, estafadores en este caso, simplificando al máximo las herramientas con las que trabajarán para crear los phishing.

Estos kits los pueden adquirir en el mercado negro. De acuerdo a la compañía de ciberseguridad Group-IB, los precios en 2019 están de media en torno a 300$. Los kits más básicos se pueden encontrar desde 20$ y los más sofisticados entorno a 900$ [3]. En el año 2019 aumentó un 149 % su precio medio respecto a 2018, cuando costaban unos 120$ [4]. Estos kits constituyen un conjunto de herramientas informáticas que facilitan toda la gestión de un phishing. Generalmente incluyen un software para crear páginas web falsas muy convincentes, y también plantillas de correo electrónico; de hecho, algunos kits más completos disponen de listas de direcciones de correo robadas y programas para automatizar la propagación del phishing a través de ellas.

2.1.2. Tipos de phishing y perfiles de las víctimas

Existen diferentes tipos de phishing, y, con ello, diferentes perfiles de víctimas. El más común es el phishing tradicional a través del correo electrónico, en el que se suplanta a una compañía conocida para robar datos sensibles, como credenciales de la víctima. Luego, existen otros tales como el *smishing* el cual se realiza vía SMS, o *el vishing* en el que el medio de contacto suele ser una llamada o un mensaje de voz (*voice – over – IP*) [5]. Otro tipo es el *malware-based phishing*, el cual adjunta en el correo un archivo malicioso o una URL que al pincharla descarga un ejecutable para introducir un malware aprovechando las vulnerabilidades del equipo del usuario [6].

Asimismo, se puede llegar a un phishing a través de sesiones de chat online, donde el estafador interacciona con otros usuarios conectados, pudiéndoles distribuir una URL maligna. Existen *banners* de anuncios falsos, de productos que puedes comprar muy rebajados que te llevan a una página donde capturan los datos bancarios. Muchas de estas técnicas buscan que compres algo, un artículo material, un software, un curso online, etc., para quedarse con esta información personal.

Sin embargo, estos tipos phishing no se lanzan siempre de forma indiscriminada, muchas veces se busca un perfil de víctima concreto. Por ejemplo, el *malware-based phishing* suele ser propagado a pequeñas y medianas empresas cuya seguridad no está del todo actualizada y tienen más vulnerabilidades. Existe otro denominado *spear phishing* cuyo ataque va también dirigido a personas, organizaciones o empresas. En este caso, la mayoría de las veces, el phisher conoce mucho mejor a la víctima: si se trata de una persona, esta suele ser un eslabón débil dentro de la empresa. Actúa del mismo modo que en el phishing tradicional, pero el mensaje está personalizado. Otro de los phishing dirigido a alguien específico es la estafa al CEO (*Chief Executive Office*) de una empresa. En este caso, el phisher se hace pasar por el CEO y envía un correo a un empleado de menor jerarquía dentro de la empresa que tenga acceso a los datos financieros para que este proceda a hacer una transacción de mucho dinero [7].

2.1.3. Métodos de engaño que utilizan las páginas fraudulentas

Cualesquiera que sean las víctimas o los métodos utilizados para contactar con ellas, en este trabajo se van a explicar algunas de las técnicas empleadas por los phishers para suplantar a las compañías mostrando una imagen de lo más convincente. Las URLs de las web fraudulentas deben parecerse a las URLs de las legítimas. Por ello, muchas de estas URLs en cuanto a su contenido y su formato, comparten características y patrones parecidos. Lo mismo ocurre con los diseños de la web, aunque pertenezcan a distintos kits de phishing, y suplanten a distintas empresas, sigue habiendo rasgos comunes que los diferencia de las páginas oficiales.

Una forma de engañar al usuario a través de la URL es utilizar letras parecidas, por ejemplo, la ‘l’ por la ‘i’ mayúscula o incluso ciertos caracteres árabes, cirílicos y hebreos para sustituir algunas

letras romanas de diseño parecido, como por ejemplo, la letra ‘a’ en romano es idéntica a la ‘a’ en cirílico, pero Unicode las traduce a caracteres diferentes [8]. Los phishers utilizan como subdominio el dominio de la compañía legítima suplantada y completan la URL con cualquier otra cadena, por ejemplo, “paypal.com.myaccount-cgibin.com”. A esta técnica se le denomina *combo-squatting* [9].

En cuanto a las páginas, hay varias formas de verificar si es un phishing. En algunos casos, la ortografía y gramática es errónea, ya que traducen texto literal de la página original de un determinado idioma a la página falsificada en otro idioma. En algunos casos, diferentes enlace como “términos de uso”, “condiciones” o “contacto” redireccionan a la misma página activa (no tiene efecto). Sin embargo, en otros, redirecciona a la página legítima de la empresa suplantada, pareciendo aún más convincente.

Esto mismo puede ocurrir cuando se cumplimentan todos los datos en la página falsa, actuando a modo de pasarela, para luego redirigirte a la página oficial. De hecho, algunos *kits* incluyen en sus pasos verificaciones con las que prueban en la página legítima si los datos introducidos por la víctima son reales o no. Esto tiene el objetivo de que el propio kit no mande al phisher datos que no son útiles. [10].

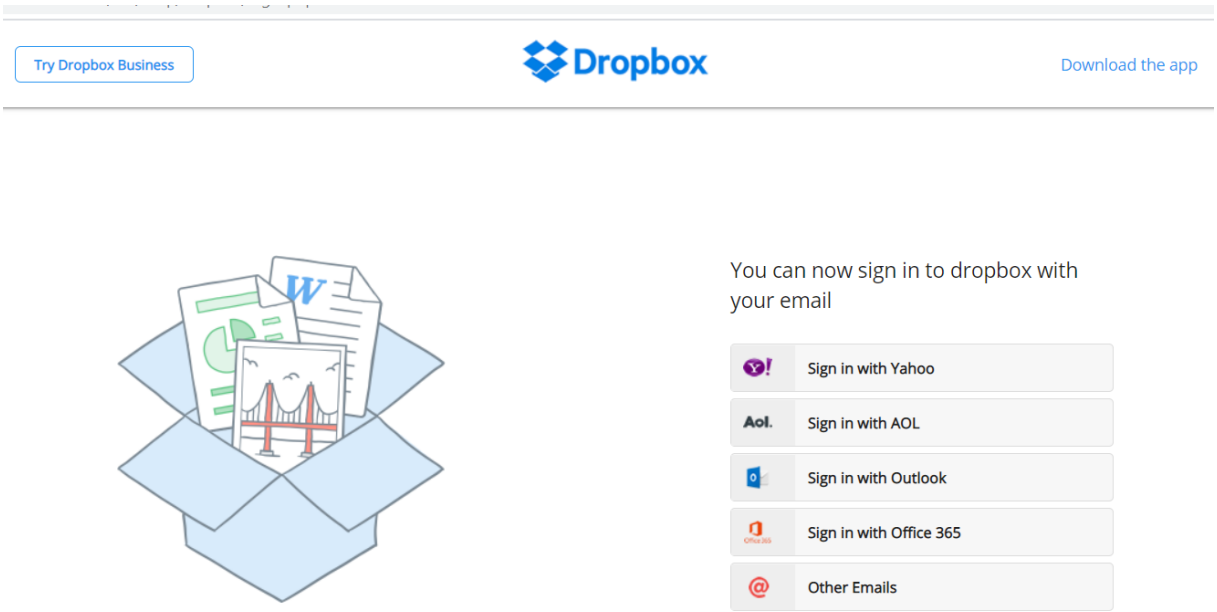


Figura 2.2: Ejemplo de página falsa suplantando a Dropbox

2.2. Evolución del phishing

El phishing surgió en torno a los años noventa y las primeras víctimas fueron los clientes de *America Online* (AOL), uno de los primeros proveedores de Internet. Los fraudes que se cometieron fueron

realizados por un grupo de ciberdelincuentes que se hacía llamar “Warez”, los cuales ya comercializaban con software pirateado. Generaban a partir de unos algoritmos, números de tarjetas de crédito aleatorias para abrir cuentas AOL y disfrutar de los servicios.

Sin embargo, *America Online* tomo medidas al respecto y consiguió tapar ese agujero de seguridad. No conformes con esto, los atacantes subieron un escalón y consiguieron falsear la web AOL, se hacían pasar por un miembro del personal y enviaban un mensaje a partir de correos electrónicos o mensajería instantánea a otros usuarios solicitando sus contraseñas y datos bancarios.

2.2.1. Ejemplos reales de phishing

En las dos últimas décadas se han reportado decenas de miles de casos de phishing. En este trabajo se muestra algunos ejemplos reales más famosos o conocidos para comprender lo realmente vulnerables que somos.

El primer ejemplo es el caso de “El príncipe nigeriano”. Se trata de un príncipe, supuesto heredero de una gran fortuna de un miembro de la aristocracia de Nigeria, el cual necesita una pequeña cantidad de dinero para completar la transacción de la herencia. En el mensaje que se manda a la víctima se dice que si colabora será recompensada con parte de dicha herencia. Aunque parezca que el contexto de este phishing sea anacrónico y poco creíble, sigue siendo una de las estafas que más se utiliza y más pérdidas provoca.

Otro caso es el denominado “Phish Phry”, calificado por el FBI como el mayor fraude internacional de phishing. Una investigación llevada a cabo conjuntamente por el FBI, *Los Angeles Electronic Crimes Task Force* y autoridades egipcias, condujeron al arresto de cien personas en Estados Unidos y Egipto. Los correos enviados por los phishers suplantaban a *Wells Fargo & Co*. y *Bank of America Corp*, logrando robar 1,5 millones de dólares de miles de cuentas bancarias.

El siguiente ejemplo se diferencia de los dos anteriores en que este phishing se trataba de un ataque muy personalizado. Es un ejemplo perfecto de “fraude al CEO”, donde el phisher falsificó el correo del CEO Walter Stephan de la empresa FACC AG y solicitó a un empleado de categoría inferior que hiciera una transferencia de dinero de 50 millones de dólares a una cuenta bancaria. El motivo que daba el correo era la financiación de un nuevo proyecto. El email fue suplantado utilizando la técnica de *spoofing* de correo. El phisher obtuvo el nombre de la cuenta de correo del Walter Stephan y le suplantó modificando en el “envelope” del correo la cabecera “From”. El empleado que recibió el correo no sospechó y procedió a realizar la transacción. Ahora se puede detectar este *spoofing* aplicando el protocolo DMARC que será explicado en el apartado 2.3.2.

2.2.2. Gráfica de la evolución del phishing desde 2004 hasta 2019

A continuación, se describen los datos que aparecen en la gráfica de la figura 2.3 que refleja cómo ha evolucionado la cantidad de reportes de phishing de correo desde el año 2004 hasta la actualidad. Estos datos se han obtenido a partir de los informes de APWG [11]. Resulta complicado explicar la progresión de la gráfica dado que habría que tener en cuenta muchas variables que se desconocen. Se citarán posibles motivos, pero que pueden haber sido totalmente intrascendentes en la realidad.

El gráfico comienza en 2004 coincidiendo con un mínimo histórico en la cantidad reportada de casos de phishing, con 13745 casos. A partir de este primer año, se puede apreciar cómo ha ido creciendo el número de casos hasta alcanzar un pico histórico en año 2009. La causa del incremento progresivo podría ser debida a varios factores: la explosión del phishing como una nueva forma de fraude muy rentable y provechosa a ojos de los estafadores; al aumento de los usuarios en la red; al desconocimiento del mundo del phishing por parte de los internautas; a las pocas medidas de seguridad utilizadas en los protocolos de correo electrónico o la falta de concienciación de este problema en las empresas y los usuarios de la red en general.

Durante los tres años siguientes hasta 2012, disminuyó el número de casos hasta en un 31 %. Algunos motivos que pudieron conducir a este enfriamiento de los ataques de phishing, podrían haber sido: mejora sustancial en las tecnologías de ciberseguridad enfocada al fraude digital, estancamiento por parte de los phishers en la creación de nuevos kits, mayor concienciación entre los usuarios de la red, etc.

Sin embargo, justo después de este periodo se muestra una subida bastante acusada, posiblemente debida a una sofisticación y mejora de las herramientas utilizadas por los phishers. Viendo la rentabilidad creciente obtenida con estas nuevas técnicas se irían sumando más y más ciberdelincuentes alcanzando un máximo histórico en 2015. Finalmente, se observa una caída bastante pronunciada, posiblemente causada por motivos similares a los ocurridos en 2009. Esta última evolución puede calificarse positivamente ya que se observan evidencias de que la tecnología aplicada para combatir ciberdelitos avanza de forma acompasada a las nuevas herramientas y kits de phishing que surgen.

2.2.3. 2020 y el coronavirus

En lo que respecta a la actualidad, año 2020, la pandemia del coronavirus (COVID-19), que está afectando a cientos de millones de personas, ha obligado a muchos trabajadores a hacer teletrabajo. El teletrabajo es una forma de trabajar telemáticamente con la que se intenta desarrollar la actividad laboral que se hacía presencialmente en el puesto de trabajo. Esto ha resultado ser una gran oportunidad para los estafadores de Internet, por lo que las empresas y los empleados deben actuar con mayor precaución, entender los riesgos de privacidad en Internet y asegurar y proteger todos los *endpoints* con los que trabajen.

Figura 2.3: Gráfico de la evolución de los reportes únicos de phishing desde 2004 hasta 2019.

Algunos de los ejemplos más claros es la herramienta multimedia de *Zoom* para reuniones y eventos telemáticos, donde mandan a la gente notificaciones sobre reuniones pidiendo credenciales y quedándose así con los datos más sensibles.

Si los phishing más efectivos son aquellos que tienen un contenido llamativo o urgente, en la actual situación de pandemia, los phishers se aprovecharán de los miedos de las personas en materia de sanidad, resultando estas más vulnerables a los engaños. El contenido de estos mensajes o estos anuncios sobre el COVID-19 hace referencia a la información más reciente sobre el virus, como métodos para combatir contra el virus, recomendaciones y facilidades, enmascarando en estos enlaces los ejecutables maliciosos.

Los phishers se las ingenian para tomar de las actividades cotidianas algunas ideas para crear un mensaje cautivador y atractivo hacia la víctima. Sin embargo, cuando ocurre una catástrofe natural como puede ser un terremoto, una alerta de tsunami, o una pandemia como en este caso, aprovechan la circunstancia para calar más en las personas consumidas por el miedo y el pánico de la situación.

La cantidad de phishing reportados en los primeros tres meses de 2020 han sido de 165.772 casos, superando ligeramente al último cuarto de 2019 [12].

En el siguiente gráfico de la figura 2.4 extraído del mismo reporte donde se han obtenido la información previa, puede observarse la proporción de la cantidad de ataques que han ido a parar a los distintos sectores: SAAS y correo electrónico, instituciones financieras (bancos, PayPal, etc.), *social media*, *eCommerce* y almacenamiento en la nube.

Figura 2.4: Proporción de los ataques de phishing a los distintos sectores en el primer cuatrimestre de 2020. Fuente: [12]

2.3. Herramientas y técnicas de detección de phishing

2.3.1. Páginas web para comprobar phishing

En este apartado se van a exponer algunas de las herramientas y técnicas que se utilizan a día de hoy para analizar páginas web y con ello identificar si responden a un phishing. Muchas de ellas son accesibles por cualquier usuario de Internet. Algunas requieren tener ciertos conocimientos básicos sobre páginas web y redes (recursos, certificados, IP, dominios, DNS, etc).

Existen páginas en Internet que permiten al usuario introducir una URL realizando un análisis de la misma mostrando una probabilidad aproximada de que sean maliciosas. Por consiguiente, no siempre van a acertar el 100 % de los casos, sino van a mostrar una buena aproximación mediante el uso de unas reglas de decisión o un proceso de inteligencia artificial más avanzado. Estas webs poseen bases de datos muy actualizadas conteniendo la identificación de URLs y archivos maliciosos, así como antivirus y motores de detección. Un ejemplo es “VirusTotal”.

Otras páginas ofrecen información acerca del propietario de un dominio o de una dirección IP así como su reputación en la red como WHOIS. Esta información se obtiene a partir de datos de reputación que pueden incluir usos de correo electrónico, recursos del sitio web, políticas y privacidad así como denuncias de usuarios sobre el nombre de dominio. Sin embargo, debido a que el RGPD (Reglamento

General de Protección de Datos) entró en vigor en el año 2018, ahora esta información permanece en su mayoría oculta.

Otras páginas como “URLScan”, además de analizar la legitimidad de una URL, muestran detalles de los distintos recursos de una web, como las direcciones IP con las que contacta, recursos estáticos como *JavaScript*, CSS, o las *cookies* que genera la página. Estos elementos son imprescindibles para hacer un análisis forense de una web. Estos mismos recursos pueden ser accedidos a través de las herramientas de desarrollo de Chrome o Firefox y otras herramientas o extensiones disponibles.

También se pueden encontrar sitios web más enfocados al phishing como “isitPhising” que utiliza inteligencia artificial, analizando más de 40 características por página. Entre estas características pueden evaluarse la forma de creación de los formularios, las redirecciones de la página (si es a la página activa, o a la página legítima, o da un error), así como unos patrones inteligentes basados en reglas heurísticas desarrolladas por *Vade Secure* para analizar rutas de archivos, *scripts* y otros contenidos estáticos de la web. Todo esto con el fin de proveer de una avanzada protección contra el phishing.

De cualquier página web se pueden obtener diferentes indicadores, se les denomina IoCs (Indicadores de Compromiso) y son necesarios para conocer qué ocurre en la red con esa página. Se emplean sobre todo en informática forense para detectar cualquier posible actividad delictiva que esté afectando a la infraestructura TI de la empresa. Un IoC es un dato o una serie de datos sustanciales o importantes, que han sido recuperados a través de análisis de los patrones de una red o un sistema operativo, que permite caracterizar un incidente de ciberseguridad como una intrusión en la computadora. Los IoCs más comunes son direcciones IP, hashes de archivos URL o malware.

Los IoCs se utilizan en sistemas antivirus para detectar futuros ataques. Es necesario que las diferentes corporaciones colaboren entre ellas para prevenirse de ataques de los ciberdelincuentes ya que en la mayoría de ocasiones esta información pasa por empresas privadas de ciberseguridad proveedoras de herramientas antivirus y otros servicios de seguridad, circunstancia que aprovechan los defraudadores, ya que dificulta la dotación de mayor agilidad de comunicación entre empresas.

2.3.2. Técnicas y herramientas para detectar phishing

Desde un punto de vista de la ciberseguridad, se puede decir que la vida de un phishing comienza en el momento en que se propaga, y acaba cuando se cierra la URL maliciosa. Las etapas son la detección, el análisis y la eliminación [13].

En la primera etapa, la detección, capturamos una URL potencialmente maliciosa. Se pueden aplicar dos tipos de detección: por vía activa y por vía pasiva. En el primer tipo, se dota a las personas y empresas con riesgo de ser engañadas, de una serie de herramientas de detección que se instalan en la propia infraestructura que se desea monitorizar con el fin de capturar posibles URLs sospechosas de phishing. El segundo caso, la vía pasiva, consiste en consultar fuentes de Internet como foros de

phishing, redes sociales, o webs con herramientas más avanzadas que aportan información constante sobre reportes de nuevos phishing.

La segunda etapa, el análisis, tiene como fin asegurar que la URL detectada como posible phishing, lo sea realmente. Asimismo, si se concluye en el análisis que es un fraude, se almacena cierta información sobre el phishing, así como los IoCs citados previamente, capturas de pantalla de la web fraudulenta, etc. Esto tiene como objetivo, identificar futuros phishing similares de forma más rápida y eficaz.

En la última etapa, de eliminación, se acometen las acciones necesarias para comunicarse con los proveedores del dominio con objeto de cerrarlo de forma permanente.

Detección activa

El protocolo SMTP (*Simple Mail Transfer Protocol*) es un protocolo diseñado en los años noventa para enviar correos electrónicos. Dentro de los protocolos de comunicaciones, es uno de los que menos modificaciones ha sufrido desde que se creó. Es un protocolo no sumamente complejo, correcto, que aborda su funcionalidad más básica. Sin embargo, la perspectiva que se tenía del uso del envío de correo en los años noventa, ha ido evolucionando a lo que es hoy en día, es decir, no está pensado para su uso masivo, así como para correos confidenciales. Esto es debido a que tiene algunas vulnerabilidades. En el protocolo SMTP se puede falsear quien es el emisor del correo y, por tanto, se puede suplantar la identidad de otra persona. Esto es una mina de oro para los phishers. En SMTP no se verifica la autenticación del emisor, es decir, no se comprueba si el emisor está autorizado para enviar un correo con la dirección de correo de la persona que dice ser.

Esto es debido a que en el envío de un correo electrónico hay dos elementos: un sobre y un encabezado. Se puede asemejar al envío de una carta física: por un lado, está el mensaje con el contenido y por otro lado, está el sobre en el que introduces dicho mensaje. En el caso del correo electrónico, el sobre con los campos “To/From” es la información que se utiliza para resolver el envío del mensaje; este sobre es el SMTP *envelope*. Por otro lado, están las cabeceras del mensaje interno. Aquí, existen otros dos campos “To/From” que muestran el emisor y destinatario auténticos del mensaje. Por ejemplo, la parte del “envelope” indica que el emisor (“From”) es “boss@gmail.com”, pero el mensaje indica que es “thief@gmail.com” [14]. En la figura 2.5 se muestra un esquema del funcionamiento del SMTP y del sobre que envuelve el mensaje de correo.

Actualmente, existen dos mecanismos de detección activa: DMARC (*Domain-based Message Authentication*, *Reporting & Conformance*), y la cabecera *referer*. DMARC es un protocolo que extiende otros dos protocolos que son SPF y DKIM. Este protocolo permite al dueño o administrador de un dominio establecer una serie de restricciones protegiendo su dominio contra un uso no autorizado, así se consigue “securizar” el dominio contra ataques *spoofing* de correo. Para ello, se especifican en los registros de DNS qué mecanismos se deben aplicar para enviar correo electrónico desde ese dominio

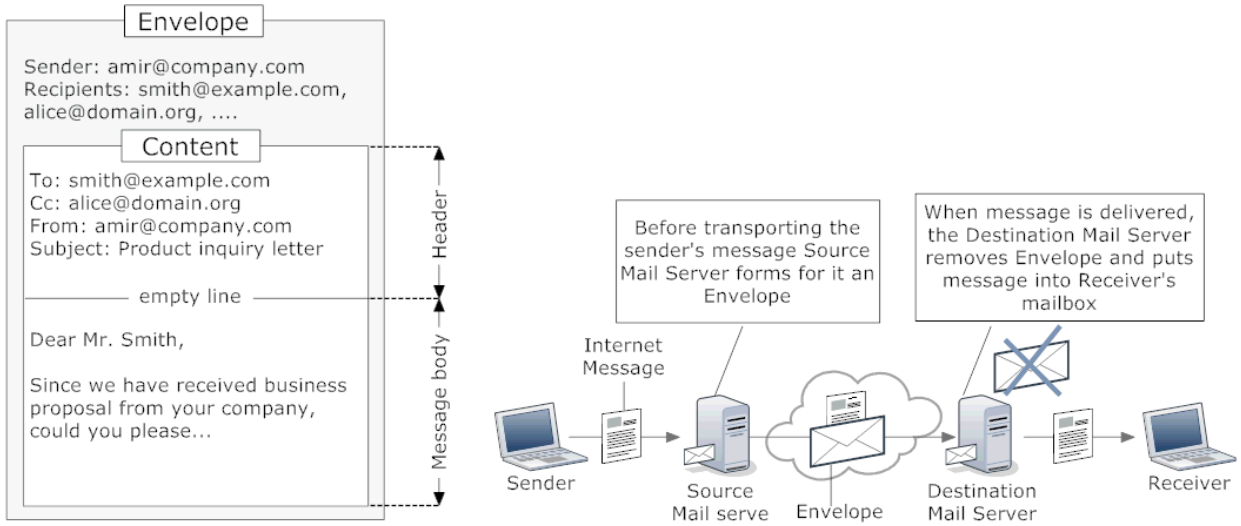


Figura 2.5: Cabecera del mensaje junto al sobre del protocolo SMTP. Fuente: [14]

y con qué información, en el caso de SPF, las IPs admitidas y en el caso de DKIM (la clave pública).

El protocolo SPF (*Sender Policy Framework*), permite especificar la política de envío de correo, por ejemplo, desde qué IP o servidores se pueden enviar mensajes. Cuando el servidor de correo receptor captura el mensaje, puede verificar si se cumple la política establecida en el dominio. Por ejemplo, en una empresa si se envía un correo con el dominio de la empresa, si la IP desde donde se ha enviado no se encuentra en la lista de rangos de IP se redirige a la carpeta de spam o directamente no se envía [15] [16]. En consecuencia, esta operativa presenta el inconveniente de que hay que actualizar los registros del SPF del DNS si se desea introducir una nueva IP desde la que poder enviar correos con el dominio de la empresa. Otro inconveniente es que este protocolo no es infalible, porque no comprueba otros tipos de *spoofing* capaces de falsear la dirección IP del correo.

Por otro lado, DKIM (*DomainKeys Identified Mail*) es un protocolo que permite a los servidores de envío de correo cifrar parte del encabezado mediante una clave privada que ha sido previamente instalada en ellos generando un hash o firma, y cuya clave pública se asigna en el registro DKIM del DNS, con el fin de que los servidores de correo receptores puedan verificar dicha firma. De esta forma, se asegura la autenticidad del mensaje. Si la parte del encabezado cifrada al aplicar la clave pública es distinta a lo que se muestra en el texto plano, es que ha sufrido alguna modificación durante la transmisión y puede ser indicativo de que haya habido un *spoofing* de por medio y por tanto se haya corrompido la integridad del mensaje [17] [18].

Estos dos protocolos, por separado, no garantizan una fiabilidad completa. Por ello, se utiliza DMARC como combinación de ambos protocolos, de manera que, si alguno falla, los servidores receptores del correo, rechazarán dicho mensaje al interpretarlo como potencialmente malicioso. Además,

existen unas direcciones de correo que se pueden asignar en el protocolo DMARC, con el fin de enviar unos informes si alguien se hace pasar por ti. Estos informes son de dos tipos: informe RUA que proporciona una información general de todo el tráfico de un dominio como, por ejemplo, el número de correos que se están suplantando, y los informes RUF que son una copia del mensaje emitido que no cumplen con las políticas de envío de correo DMARC [19].

Sin embargo, estos reportes, hoy en día, no pueden emitirse debido a que se puso en marcha las RGPD, y debido a estos problemas de privacidad, la mayoría de proveedores de informes DMARC no proporcionan informes RUF [20]. Además, muchas entidades no los solicitan para evitar responsabilidades futuras. Al final, DMARC se instala con el fin de obtener estadísticas del dominio y para detectar si se hacen suplantaciones de identidad por correo electrónico.

Existe otro mecanismo, como se citó previamente, que está basado en la cabecera “referer”. Los phishers necesitan que sus víctimas confíen plenamente en la página falsa que están visitando. Para ello, una técnica que emplean es engañar a las víctimas redirigiéndolas a la página legítima, una vez que han ingresado sus datos. Evidentemente, les saldrá la página de registro o de acceso de nuevo, como si hubiese habido algún error en la petición, pero ahora estarán en la legítima, entenderán que habrán cometido algún fallo al introducir las credenciales y rellenarán el formulario de nuevo.

Y es por esto que “referer” resulta tan útil, puesto que consiste en una cabecera de la que disponen las peticiones HTTP realizadas desde una página para identificar la URL de la que se originó la petición. Esto ofrece una gran ventaja desde la perspectiva de la ciberseguridad ya que, si los dominios no coinciden, o el dominio del que se ha venido no aparece en el *ranking de Alexa* (*ranking* establecido a partir del tráfico que tienen las páginas) o no es muy conocida, es un indicativo de que se está cometiendo un phishing. Esto se puedo monitorizar comprobando en los logs del tráfico entrante las cabeceras de las peticiones.

Detección pasiva

La detección pasiva consiste en buscar información de phishing recurriendo a distintas fuentes en Internet. Foros como “PhishTank”, son páginas colaborativas donde la gente reporta casos potenciales de phishing e información relacionada, donde además pueden calificar si la consideran o no phishing. Otras páginas incorporan inteligencia artificial en sus análisis para identificar casos de phishing, como “OpenPhish”. Como se mencionó previamente “URLScan” facilita las búsquedas de phishing permitiendo incluir distintos filtros que modifican las cadenas URL (por dominio, etc).

También se encuentra información relacionada en fuentes abiertas tales como Google, Twitter, foros especializados de phishing u otras redes sociales. Otras compañías como “NETCraf”, proveedora de servicios de seguridad de Internet, suministran a los principales motores de búsquedas como Google (Chrome, Gmail, etc.), Firefox, Opera, Safari, de análisis automatizados en tiempo real de las páginas consultadas para evitar que el cliente acceda a recursos maliciosos. Esta facilidad permite elaborar

unas listas negras o filtrado web que recopilan dominios sospechosos o identificadores de URLs maliciosas. El inconveniente está en que siguen permaneciendo vigentes y siguen teniendo riesgo potencial de que alguien pueda acceder a ellas y acabar siendo engañado.

3

Arquitectura antiphising

Partiendo de una visión global del phishing, se plantea una solución escalable siguiendo la misma línea de investigación, mejorando la arquitectura creada para así, afinar más en la reducción del impacto del phishing. Se ha comprobado en los apartados anteriores que se trata de un problema que va a persistir durante los años, ya que la ingeniería social no es un error o una vulnerabilidad técnica que pueda solventarse con alguna herramienta informática, sino que se trata de la propia naturaleza humana. Por ello, la ciberseguridad tiene una gran responsabilidad y aún mucho trabajo por delante.

El objetivo principal de este trabajo es crear una arquitectura base que reciba tantas URLs por minuto como sea posible y lleve a cabo el análisis de todas ellas para identificar cuáles son legítimas y cuáles son susceptibles de ser calificadas como phishing y a qué empresas afectan (suplantan). El sistema está basado en dos procesos independientes.

El primer proceso, consiste en la dotación de nueva información a la base de datos de páginas web que se conocen que son legítimas o ilegítimas y que interesan incorporar al sistema para hacer cada vez más sofisticados los análisis. El segundo proceso consiste en la recepción de URLs de páginas que son potencialmente maliciosas, pero que no se sabe con certeza si son o no phishing. Estas URLs analizadas no sirven para realimentar el sistema con nueva información, sino que son simples análisis de los recursos web de la página cotejando su información con la almacenada en la bases de datos. En un principio, se pensó en hacer una combinación de ambos, que consistía en que las consultas que se hiciesen al sistema sirviesen además para actualizar la bases de datos e introducir nueva información, pero a efectos prácticos, no resultaba eficiente para los análisis ya que se metía redundancia y se perdía información útil.

Primer proceso: almacenamiento de información

Respecto al primer proceso, para hacerlo efectivo y funcional, es necesario ofrecer al sistema la capacidad de almacenar información clave de la URL de entrada, como una captura de pantalla de la página web o una serie de IoCs que serviran para ir mejorando los análisis posteriores. Por ello, el software dispone de dos plataformas a las que se pueden mandar URLs para recoger información. La primera de ellas es el cliente web y la segunda es la API.

El cliente web está pensado para un trabajo más manual como el almacenamiento puntual de una URL concreta o un hash específico que se conoce que es legítimo o ilegítimo. Por la parte que corresponde a la API, está pensada para recibir a través de procesos automatizados gran cantidad de URLs de las que también se conoce si corresponden a las páginas oficiales o a páginas falsas. Estos procesos automatizados que capturan estas URLs y las envían al sistema no son requerimientos para este TFG.

Segundo proceso: consultas

En cuanto al segundo proceso, no tiene sentido realizar análisis sobre URLs si el sistema no tiene la capacidad de recibirlas. Por ello, el software dispone de las mismas plataformas comentadas en el primer proceso (cliente web y API) a las que se pueden mandar URLs para analizarlas. El cliente web está pensado para comprobar puntualmente una URL. Por ejemplo, queremos verificar si una URL que nos ha llegado a nuestro correo electrónico es phishing. Para ello se copia y pega la URL en el buscador de la página web que dispone el software y este procede a su análisis.

Respecto a la API, igual que pasaba con el primer proceso, está pensada para recibir gran cantidad de URLs de distintas fuentes. Recuperarían URLs de forma automatizada de las fuentes abiertas y otras herramientas que se mencionaron en el apartado 2.3.2 como PhishTank o URLScan, y se enviarían al sistema a través de la API para ser analizadas.

3.1. Evolución del software

El proceso de creación del software se ha planteado en dos fases. La primera contempla la investigación del problema planteado y la búsqueda de una solución informática. La segunda fase consiste en la implementación de la solución, habiéndose organizado en tres iteraciones.

El trabajo de investigación consistía en recopilar metodologías utilizadas por los estafadores para construir sus páginas web falsas, con el objetivo de diseñar un sistema lo más optimizado posible que fuera capaz de analizarlas y concluir si la página en cuestión se trataba de un phishing o no.

En lo correspondiente a las iteraciones, la primera tenía el objetivo de crear todas las funciones de análisis y de almacenamiento, así como crear la base de datos y las consultas pertinentes. La segunda iteración consistía en crear el cliente web y la API, para poder hacer unas primeras pruebas que simularan un entorno real o de producción. Finalmente, en la tercera iteración se plantea trabajar con la herramienta Docker y crear un sistema distribuido en el que haya un sistema de colas y nodos de forma que permita una escalabilidad horizontal, esto es, aumentar la capacidad del sistema incluyendo tantos nodos como hagan falta para abarcar el flujo de URLs que pasan por el sistema.

En los siguientes apartados se explica cada una de estas fases e iteraciones. Conviene consultar

en paralelo el apartado 3.2 donde han sido explicados con mayor profundidad algunas herramientas que se mencionan.

3.1.1. Trabajo de investigación

Antes de comenzar a diseñar el software, se tuvo que hacer un trabajo previo de investigación sobre el phishing para entender bien el contexto, *el background* y el *modus operandi* de los phishers con el objetivo de empezar a buscar una forma de analizar las páginas web en base a ello. Esta investigación consistía principalmente en recopilar todos los métodos de engaño que utilizan los phishers para suplantar a una entidad a partir de una página web fraudulenta. Así pues, a partir de esta información se buscaba construir un sistema software tal que dada una página web, pudiera identificar cada uno de estos métodos, y a partir de unas reglas heurísticas, clasificar si era phishing o no y a qué entidad suplantaba.

Muchas de las trampas o metodologías utilizadas por los estafadores que se tomaron como relevantes, están descritas en el apartado 2.1.3, por ejemplo, el formato de la URL (los caracteres utilizados, los subdominios), el tipo de recursos de la página o qué redireccionamiento se utilizaban en ella, etc. Sin embargo, aunque no se descartara en ese momento de la investigación ninguna de estas metodologías, se llegó a la conclusión de que una buena forma de enfrentarse al problema y analizar las páginas era a partir de los IoCs (los indicadores de compromiso que ya se han explicado en otros epígrafes de la memoria, 2.3.1).

Y es que en primera instancia, los hashes (una forma de IoC) iban a facilitar realmente el trabajo de análisis y comparación de recursos web de las páginas de phishing.

A partir de ahí, se formalizó el proceso de la construcción del software. En un principio, no se conocían el número de iteraciones que se iban requerir, sino que a medida que se fueran cumpliendo hitos, evolucionando y completándose el software se asignarían nuevos objetivos, por lo que no ha habido un proceso de ingeniería de software previo a la construcción del mismo que haya marcado la trayectoria del proyecto desde un inicio.

3.1.2. Primera iteración

Los primeros pasos que se han dado en la implementación de este software, han ido orientados a la funcionalidad más básica y trascendente en este proyecto. Esta primera iteración cubre todas las funciones encargadas de capturar los recursos de una web y aplicar a cada uno la función hash SHA-256, las funciones de análisis y todo lo referente al módulo de la base de datos: la creación de las tablas y las consultas pertinentes. Para entender cómo ha sido diseñada la función de análisis de una página, puede consultarse el apartado 3.1.2, al final de esta sección.

Existe una herramienta pública que se utiliza para capturar todos los recursos de una web, llamada “browser-mob proxy”, que ofrece al programador la posibilidad de capturar las peticiones HTTP, es decir, las respuestas del servidor donde se aloja la página web y exportar dichos datos en un formato HAR file. Con ello, se captura cada uno de los recursos de la web y se filtran los que realmente interesan, es decir, aquellos que hacen única a la página web. Esto se debe a que existen recursos que dependen de scripts y que cada vez que se accede a la página cambian, por lo que no resultan útiles para nuestros análisis; u otros recursos estáticos como tipografías, CSS, o *JavaScripts* que no aportan valor porque no son exclusivos de una página sino que son comunes a muchas páginas web. Una vez se escogen los recursos capturados para el análisis, se les aplica la función hash SHA-256 para obtener un identificador único de los mismos.

Cada uno de estos hashes o identificadores se almacena en la base de datos, y servirán como fuente primaria para los futuros análisis de las URLs entrantes. La base de datos está creada bajo el sistema de gestión de bases de datos SQLite. Consiste en un fichero en Python en el que se define una clase DBHandler, y en ella se crea toda la configuración de la base de datos a partir de la libreria *sqlite3*. Esta configuración contempla la conexión a la base de datos, la creación de las tablas, las consultas y otras fuciones complementarias etc.

Se llegó a la conclusión de que este fichero de Python iba a ser la mejor alternativa para gestionar el almacenamiento. Inicialmente se barajó la posibilidad de implementar un servidor MySQL, pero dada la funcionalidad tan minimalista del software, finalmente, se optó por implementar una base de datos a partir de este fichero, pues resultaba más sencillo, y práctico. Se me facilitó un esquema básico de este fichero el cual contemplaba la creación de la conexión con la base de datos SQLite, junto a una tabla de ejemplo y un par de consultas básicas de ejemplo. A partir de esto, se revisó el esquema y se adaptó de acuerdo a la funcionalidad que se quería tener hasta obtener como resultado el fichero Python final en el que se gestiona todo el almacenamiento del sistema.

Selección de hashes

Los hashes que están almacenados en la base de datos han sido seleccionados a partir de dos procesos: uno pseudo automático y otro manual.

Por un lado, en el proceso pseudo automático interviene un campo en la base de datos que denominamos “matches”. El objetivo de este es dar más valor a los hashes de los recursos más comunes entre kits de phishing que suplantan a una compañía concreta. El proceso sería el siguiente: partimos de un conjunto de páginas webs que sabemos que son fraudulentas; se aplican las funciones de análisis sobre estas páginas, y se va sumando una unidad a aquellos recursos que se van repitiendo. Por ejemplo, la imagen que corresponde al logo de Dropbox tiene su hash, y, por tanto, las distintas páginas web que tengan exactamente ese mismo recurso tendrán el mismo hash, por lo que se sumarán tantas unidades como veces aparezca dicho hash. De hecho, tendría sentido que el hash del recurso

web correspondiente al logo de Dropbox sea de los que más matches tengan.

Esto tiene la desventaja de que haya recursos con muchos “matches” debido a que aparezcan en múltiples páginas web de diferentes compañías. Sin embargo, resultarían inútiles para nuestros análisis, pues no aportarían exclusividad de una compañía. Por ello, se ha propuesto eliminar de la base de datos aquel hash que se repita para compañías diferentes.

Por otro lado, interesa que puedan seleccionarse e insertarse hashes de forma manual, debido a que existen algunos recursos de páginas falsas que las personas expertas en fraude pueden valorar como muy relevantes sin necesidad de llegar a esa conclusión a través de un proceso automático. Son recursos que hacen únicas a estas páginas y ayudan a identificarlas rápidamente. Es evidente, que aunque la informática facilita la automatización de procesos, el factor humano sigue siendo de vital importancia.

Función de análisis

El análisis de una página web corresponde al segundo proceso mencionado en la introducción del capítulo 3. Este segundo proceso busca analizar una web a partir de los recursos capturados de la misma y la información almacenada en la base de datos. En el anexo C.2 se muestra el código en Python de la función encargada de hacer este análisis. Es importante haber consultado previamente al apartado 3.2.2 donde se explican las tablas de la base de datos y sus campos para entender cada una de las variables utilizadas.

Previo a la realización de dicho análisis, se habrán capturado los recursos web de la página solicitada, habiéndose almacenados en forma de diccionario y se habrá invocado a esta función pasándole como argumento dicho diccionario. Este diccionario aparece en el código bajo la variable “dic”. Las claves del diccionario son los hashes del recurso y el valor es la URL del recurso. El diccionario de la solución es el llamado “dic_results”.

Por cada hash se hace una consulta a la base de datos para ver si ese hash coincide con algún otro hash almacenado en ella. Si es así, obtenemos el nombre de la compañía a la que está ligada el hash. Recordamos que no es posible que un hash pertenezca a diferentes compañías, porque no sería información útil para el análisis. Asimismo, si resulta efectiva la consulta, se obtienen el número de “matches” de ese hash y la legitimidad del mismo. Los hashes que aparecen en la base de datos junto a su información (“matches” y legitimidad) constituye nuestra información a priori. Es posible que el hash del recurso web que se esté analizando pertenezca a una web legítima, y sin embargo, la información que aparece en la base de datos de este hash provenga de una página fraudulenta. Por tanto, en los resultados ese hash clasificará la página web como fraudulenta, cuando realmente es legítima.

Si finalmente tras hacer la consulta definida en la linea 14 del código, no obtenemos ningún hash

coincidente, se suma una unidad a “others”, donde se guarda el número de hashes de la página web consultada que no aparecen en la base de datos. Por el contrario, si existe, se actualizarán una serie de valores en el diccionario de resultados.

En general, para todos los hashes que han “matcheado” con alguno de la base de datos se guardan los siguientes valores por cada compañía: el número de “matches” de hashes totales, el número de hashes de páginas legítimas, el número de hashes de páginas fraudulentas, y el número de hashes que coinciden en ambas páginas (esto es, hashes que han aparecido tanto en las páginas oficiales como en aquellas que son fraude). Asimismo, se almacena el hash y la URL del recurso en el diccionario de resultados. En el anexo E.2 pueden observarse un par de ejemplos de consultas de una URL donde se muestran cómo aparecen todos estos valores en el frontal web.

3.1.3. Segunda iteración

Una vez hechas las pruebas de la base de datos y las funciones encargadas de la captura de los recursos con la herramienta “browser-mob proxy”, el siguiente cometido es construir un cliente web con el que hacer peticiones introduciendo una URL directamente en el buscador de la página (simulando un entorno de producción) y visualizar los resultados.

Asimismo, se ha creado la API, cuyo objetivo es permitir la comunicación entre diferentes software. En este caso, los tipos de software se refieren a los procesos automatizados que recuperan URLs maliciosas y al sistema de análisis creado. Así se consigue que no haya necesidad de que estos procesos conozcan la implementación del sistema de análisis. Una vez construida la API, la parte de la aplicación web se adaptó de forma que las vistas no implementaran nada de la parte funcional del análisis, sino que redireccionaran la petición del cliente web hacia la API. De este modo se evitaba redundancia en la implementación, optimizando y simplificando el software, de manera que la vista esperaría a recuperar el resultado de la API y solo tendría que renderizar la plantilla correspondiente con los resultados obtenidos. En el anexo E.2 se muestran algunas de las vistas de la aplicación (frontal web).

El diseño de la parte estática de las plantillas se ha construido a través de una herramienta llamada Boostrap. Las tablas, la cabecera y los formularios han sido creados a partir de diseños que se pueden encontrar en la web de *Boostrap*, [21]. Se trata de un conjunto de técnicas avanzadas de CSS, a las cuales se puede acceder referenciando a la URL donde se aloja dicho CSS. Después, sólo es necesario copiar y pegar estructuras ya creadas en el propio código HTML y manipularlas ligeramente para adaptarlas a la distribución correcta diseñada para la web.

Para construir el frontal web se ha empleado el framework de Flask [22]. Ya que existía flexibilidad para construir el software entre Django y Flask, se eligió Flask. Django es más robusto y está preparado para crear aplicaciones web complejas con más facilidad y de forma más rápida. Flask, está pensada

para sitios web más simples, pues su diseño es más minimalista; además, resulta más comodo utilizar Flask para implementar la API.

La API se ha creado con la herramienta Flask Blueprint, su funcionalidad es independiente a la aplicación Flask principal. Sin embargo, es necesario registrar en el código de la aplicación ("app.py") la API para que la aplicación Flask extienda su contenido y trabajen conjuntamente.

Eficiencia con hilos

El análisis de una web requiere de dos acciones que no resultan ser precisamente rápidas, estas son: la captura de los recursos de la página web y la captura de pantalla de la misma. No es posible mejorar los tiempos de forma individual de cada una porque depende de factores externos como son la velocidad de Internet, o el tiempo de respuesta de propia herramienta "browsermob-proxy", que es la responsable de capturar los recursos web. Por tanto, la solución que se ha barajado para aligerar el análisis es la utilización de hilos, de forma que se pueda ejecutar en paralelo tanto la captura de los recursos como la captura de pantalla.

La implementación se basa en una función "thread_screenshot" en la que el proceso funciona de forma indefinida en un bucle "While" y dentro de este bucle, se queda esperando en otro bucle "While" hasta que se activa un flag, comenzando con la captura de la pantalla. El proceso principal, se encarga de capturar los recursos y aplicar el hash. La comunicación entre el hilo y la ejecución principal se hace mediante variables globales. La funcionalidad de este hilo puede verse en el código 3.1

Código 3.1: Código correspondiente a la función que realiza el hilo.

```
1  def thread_screenshot():
2
3      global URL_global
4      global img_global
5      parser1 = Parser(logger)
6      parser1.iniciarConexionChromeDriver()
7
8      while (True):
9          while (URL_global == None):
10             pass
11         URL = URL_global
12         try:
13             img_global = parser1.take_screenshot(URL)
14         except:
15             img_global = None
16         time.sleep(0.005)
17         URL_global = None
18
19 y = threading.Thread(target=thread_screenshot)
20 y.start()
```

3.1.4. Tercera iteración

En la tercera iteración se plantea mejorar y completar la arquitectura base de forma que sea sencilla de implementar en cualquier equipo, con cualquier tipo de sistema operativo y que, además, tenga la capacidad de ser escalada horizontalmente. Para ello se utilizarán dockers.

Docker es una tecnología basada en contenedores utilizados para crear entornos virtuales en los que poder correr una aplicación o un software. Estos dockers se ejecutan sobre el núcleo de la máquina host y resultan ser mucho más ligeros, livianos y eficientes que una máquina virtual. Además, se trata de un framework que integra múltiples funcionalidades para ofrecer todo tipo de facilidades a los desarrolladores para montar sus aplicaciones y sistemas distribuidos.

Para hacer mucho más compacto el software y que permita su migración o instalación en cualquier otro equipo o máquina se optó por utilizar esta tecnología. Todos los detalles sobre la implementación y el funcionamiento de Docker puede consultarse en este apartado 3.2.5.

Teníamos dos objetivos consecutivos a la hora de trabajar con Docker. En primera instancia, se buscaba hacer funcionar el sistema, al menos, en un único contenedor a través de un *Dockerfile* y el segundo, conseguir crear un sistema de contenedores que facilitara la escalabilidad del sistema y la mejor gestión del mismo a través del *docker-compose*. Ambos han sido trabajados, pero como se explicará en el capítulo 4 “Conclusiones” este segundo objetivo no ha sido cubierto finalmente.

3.2. Elementos y herramientas del software

El objetivo de este apartado no es tanto explicar los motivos del diseño, puesto que ya se han mencionado en el apartado de la evolución del software, sino describir de forma detallada la parte más técnica así como la implementación y composición de cada una de las herramientas y elementos utilizados en el software.

3.2.1. Hash

La palabra clave de este trabajo es “hash” ya que el funcionamiento de este software gira en torno al hash. Este consiste en una secuencia de caracteres alfanuméricos que tienen una longitud fija dependiendo del algoritmo criptográfico aplicado. Esto sirve para identificar a un recurso de forma única, permitiendo así comprobar siempre su integridad y compararlo con otros hashes para cualquier otro fin.

La función hash criptográfica no es una función biyectiva, sino que es unidireccional. Esto significa que no se puede obtener el contenido original del archivo al que se le ha aplicado el hash a partir del propio hash. Estas funciones se pueden aplicar a cualquier tipo de archivo o recurso.

En este caso, para el contexto en el que trabaja el software, interesan solamente los siguientes tipos de recursos web: png, jpg, gif, tiff, svg e ico. Otros tipos como css, js, json han sido rechazados, como ya se ha llegado a comentar anteriormente. La explicación del funcionamiento de los análisis a partir de los hashes está recogida en el apartado 3.1.2.

3.2.2. Bases de datos

El almacenamiento es imprescindible para llevar a cabo un control de la información que obtenemos de las páginas web y de los kits de phishing para recurrir a ella en el futuro. La herramienta que se ha utilizado para resolver dicho almacenamiento es una base de datos relacional SQLite.

La base de datos diseñada está compuesta de dos tablas. La primera de ellas es la de “compañías” y la segunda “hashes”. En la primera se alojan todas las compañías de las que queremos hacer un seguimiento. El fin a largo plazo es almacenar cualquier empresa que pueda ser suplantada y así poder monitorizar cualquier entidad privada que tenga riesgo real de ser afectada por el phishing.

En la segunda tabla se almacenan los hashes de los recursos web de las páginas de phishing. Antes de continuar con la explicación conviene aclarar un par de aspectos: una empresa puede ser suplantada por kits de phishing diferentes y un kit de phishing te permite asimismo suplantar a múltiples empresas. En este trabajo, no se busca diferenciar entre kits, sino entre compañías. Esto es, que una compañía puede almacenar hashes de numerosos kits de phishing, pero el resultado del análisis, no va a indicar el *kit de phishing* que se ha utilizado para suplantar a la compañía, sino indicar directamente la compañía suplantada.

Por ejemplo, tenemos almacenado en las bases de datos hashes de recursos de páginas falsas que suplantan a PayPal de tres kits de phishing diferentes. Cuando entra una nueva página para el análisis, se comparan los recursos de esta página con los recursos almacenados en la base de datos. Si la comparación es efectiva, se devuelve como resultado que la página consultada suplanta a PayPal. En la figura 3.1 se muestra el esquema de las tablas de las bases de datos con todas los atributos.

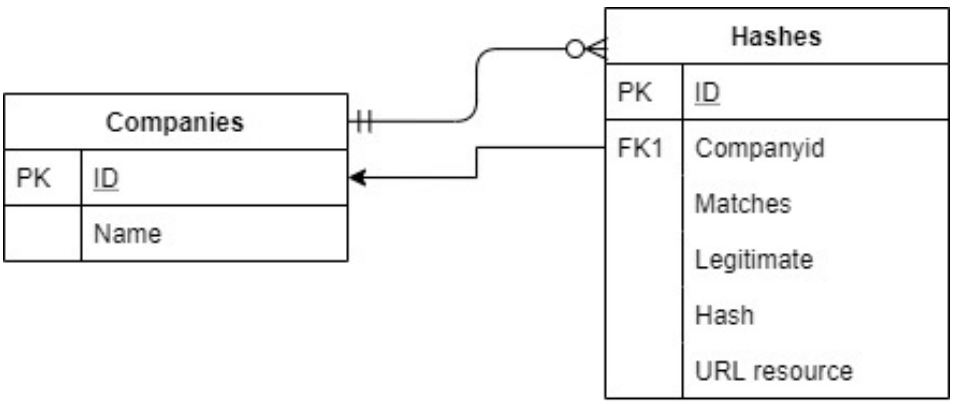


Figura 3.1: Esquema de la base de datos.

Los hashes que están almacenados en la base de datos son escogidos a través de un proceso manual y automático. Ha sido explicado con más detalle en el apartado 3.1.3.

3.2.3. Frontal Web

Como se explicó en la evolución del software, el frontal web esta construido a partir del framework de Flask. Dispone de tres rutas: “index”,“insert_hash” y “insert_url”. Son las dos únicas rutas que necesitamos para probar toda la funcionalidad del sistema como cliente web.

La primera se encarga de renderizar la plantilla correspondiente a la página principal o “index”, es decir, la plantilla que dispone de la barra de búsqueda donde se introducirán las URLs.

Una vez que introducimos una URL, se muestra el resultado de la ejecución, para ello se renderiza la misma página, pero con los resultados del análisis. Para ello se utiliza una herramienta llamada “Jinja2” la cual es capaz de generar contenido HTML de forma dinámica. *Jinja* requiere emplear una sintáxis concreta y es a través de una serie de variables como se introducen todos los datos en la plantilla que se va a renderizar.

La segunda ruta “insert_hash” renderiza la plantilla correspondiente al formulario del hash de la página web. Este formulario se utiliza para insertar en la base de datos, concretamente en la tabla hashes, un nuevo hash para la compañía. Si se desea insertar una nueva compañía, solamente haría falta introducir el nombre de la misma en el campo de “Compañía” y opcionalmente un hash.

Una vez que se cumplimenta el formulario y se envía, el servidor devuelve un resultado mostrándose en forma de tabla. Este resultado recoge todos los hashes almacenados de la compañía indicada en el formulario de envío. Esto incluye los hashes que ya había almacenados y el nuevo hash.

De la misma forma, en la ruta “insert_url” se inserta el hash de cada recurso de una URL, y el servidor devuelve también la tabla con los antiguos hashes y los nuevos hashes almacenados.

3.2.4. API

La API está creada con la herramienta de *Flask Blueprint*. La funcionalidad es independiente a la aplicación Flask, pero es necesario registrarla en la aplicación para que pueda funcionar, código 3.2. Se registra por el prefijo de URL por el que se quiere acceder, en nuestro caso: ’/api/...’. La API está constituida de cuatro rutas. Las tres primeras se diferencian de la última en que, son simples consultas a la base de datos.

Una de estas tres rutas es: “/companies/”. Se trata de una simple petición GET con la que obtenemos una lista de todas las compañías con sus identificadores respectivos. Además, podemos concatenar al subdirectorio de “companies” un id directamente para consultar los hashes de una compañía, se

Código 3.2: Registro de la API a la aplicación.

```
1 app.register_blueprint(api_bp, url_prefix='/api')
```

trata de la segunda ruta de la API. Asimismo, este id puede pasarse en la petición mediante un objeto JSON, que puede ser el id o el nombre de la compañía.

La tercera y la cuarta ruta corresponden respectivamente a "/insert-hash/" y "/insert-url/", cuya funcionalidad ya se ha descrito en el apartado 3.1.3.

Finalmente, la ruta más trascendente en lo que se refiere a los fines del software es la de "/search/". Al principio, cuando se trabajó con el *docker-compose*, se implementó dicha vista de forma que se conectaba a un servidor Redis para encolar la nueva petición. Debido a las dificultades encontradas y al escaso margen de tiempo para la entrega del trabajo, se decidió eliminar toda la parte referente al Redis, y se simplificó la función a como está actualmente.

3.2.5. Docker

Hasta este momento, el software que hemos creado era una aplicación desarrollada en el entorno de programación *PyCharm*. Este incluye un entorno virtualizado de Python en el que se instalan todas las librerías y dependencias necesarias para hacer funcionar la aplicación. Como entorno de desarrollo ofrece muchas ventajas brindando servicios integrales al programador en el desarrollo de sus aplicaciones. Facilita la creación de código con apoyo de múltiples herramientas que ofrece el entorno. Sin embargo, para trasladar el software creado a cualquier máquina o equipo para ponerlo en funcionamiento, requiere de una serie de configuraciones, instalaciones y requisitos, que dependiendo del sistema operativo subyacente puede acabar dando complicaciones. Es decir, es muy probable que acaben fallando, sea debido a la distribución del sistema operativo, de las versiones, de compatibilidades, etc. Para ello, una alternativa muy eficiente y fácil de usar es la tecnología que está despuntando llamada Docker.

Lo más parecido a un entorno de virtualización para hacer funcionar las aplicaciones en los servidores compartiendo el hardware eran las máquinas virtuales las cuales quedaban aisladas del sistema operativo "Host". Una nueva tecnología bautizada como Docker y lanzada en 2013 resultó ser más ligera, más rápida y optimizada. Actualmente, está en auge el uso de los llamados "contenedores de software". Se trata de un producto compuesto por un conjunto de elementos que conforman un entorno de ejecución de aplicaciones sin necesidad de incluir un sistema operativo dentro del mismo, dado que se apoyan en el de su anfitrión, lo que los hace muy livianos, eficientes, y de instalación mucho más sencilla y menor coste que por ejemplo las máquinas virtuales.

Todo esto tiene la ventaja de que, si cambiamos la máquina host donde queremos ejecutar una

aplicación, ésta siga funcionando sin provocar los frecuentes fallos y errores derivados de ese cambio de contexto de ejecución, lo que, por otro lado, facilita la escalabilidad de las aplicaciones. Tan solo será necesario tener instalada la herramienta Docker en aquellos equipos en los que se quiera instalar una aplicación que está pensada para ejecutarse en un entorno concreto.

Su funcionamiento no es el mismo que el de las máquinas virtuales, pues éstas precisan de un sistema operativo instalado en su entorno virtual para que puedan ejecutarse las aplicaciones. En los contenedores, el sistema operativo está fuera, es el del host que alberga el contenedor, funcionan en un nivel o plano más bajo que el de las máquinas virtuales.

Los contenedores de software actúan a modo de burbuja, encapsulando en su interior el código del software que se desea incorporar dentro del contenedor, junto con las herramientas y elementos que lo acompañan, como las librerías y otros ficheros. Además, la definición de un contenedor permite concretar qué información y recursos de la máquina (CPU, memoria . . .) donde está alojado estarán disponibles, lo que facilita enormemente la gestión de estos recursos por parte de los administradores de sistemas.

Por otro lado, a medida que el número de aplicaciones y contenedores va creciendo se hace indispensable contar con un gestor que administre un número tan grande como resultado de esa escalabilidad, pues hacerlo manualmente sería muy costoso en tiempo y dinero.

En la figura 3.2 obtenida de la página oficial de Docker, se muestra en el diagrama cómo funciona la abstracción del Docker frente a las máquinas virtuales.

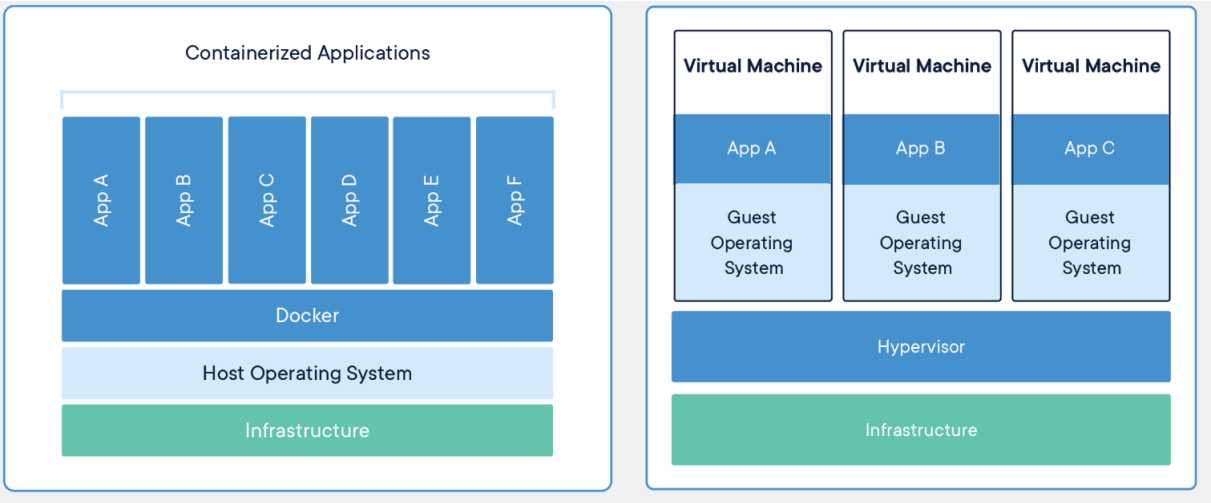


Figura 3.2: Diagrama de funcionamiento de Docker frente a las máquinas virtuales sobre el sistema operativo host. Fuente: [23].

Para tener una información mucho más detallada de Docker puedes acceder a [23] o [24], donde a parte de explicar las ventajas y su funcionalidad, dispone de tutoriales para construir desde cero un un contenedor.

La decisión principal para utilizar Docker fue la escalabilidad. Se pensó en tener una arquitectura distribuida, que permitiese ampliar su capacidad añadiendo tantos nodos como hicieran falta para un correcto funcionamiento de acuerdo a las peticiones entrantes de URLs.

Docker dispone de dos tipos de archivos de configuración para construir estos entornos virtualizados. Estos son *Dockerfile* y *docker-compose*.

El primero está pensado para definir las características de un tipo de instancia o proceso, en él se precisa la imagen en el que correrá la aplicación, así como todas las librerías, las dependencias, las macros, los permisos, la distribución y rutas de carpetas que tendrá el nuevo entorno. Esto se consigue a partir de una serie de instrucciones definidas como macros. Algunas de las más utilizadas son FROM, RUN, ENV y CMD. Todos los tipos de instrucciones: sus significados, así como para qué funcionan pueden ser consultadas en [25].

El segundo tipo, el *docker-compose* permite la creación de varios contenedores, permitiendo la ejecución de varias aplicaciones o herramientas simultáneamente dando la posibilidad de que exista comunicación entre ellas por los puertos gracias a que comparten la misma red interna. Cada uno de estos contenedores es un servicio, y pueden o no tener su propio *Dockerfile*. Esto significa, que no todas las imágenes asociadas a los contenedores requieren de un *Dockerfile*, puesto que la propia imagen ya implementa toda la funcionalidad requerida. Como se ha comentado en el apartado 3.1.4, se buscó implementar el *docker-compose*, pero finalmente, debido a algunas complicaciones en el desarrollo, no se acabó de implementar. Aun así, para que sirva de referencia y de ejemplo, en el anexo D se muestra una versión (todavía en pruebas) del *docker-compose* con el que se buscaba crear una arquitectura aún más sofisticada.

Por ejemplo, el servicio de Redis del *docker-compose*, motor de bases de datos en memoria, nos iba a facilitar la gestión de las colas. Toda la funcionalidad que requiere esta parte de la arquitectura viene en la propia imagen, no es necesario recurrir a un *Dockerfile* para implementar otros recursos o herramientas. Por el contrario, en el caso del servicio web, sí que requiere de un *Dockerfile* ya que la imagen utilizada no cuenta con algunos detalles que requerimos para nuestra aplicación. A continuación, en el código 3.2 definimos el contenido de este archivo de configuración.

Finalmente, lo que se ha acabado desarrollando es un contenedor individual con toda la funcionalidad del sistema (bases de datos, frontal web y análisis). Se van a explicar el conjunto de instrucciones, para dar una idea más clara de la configuración del docker web, definido en el *Dockerfile*.

En primer lugar, la instrucción FROM hace referencia a la imagen base en la que va a correr nuestro software. En este caso nuestra imagen utilizará “joyzoursky/python-chromedriver:3.7-selenium”. Se trata de una imagen obtenida del repositorio de Docker Hub. Docker Hub al igual que otras plataformas de código abierto donde cada usuario sube implementaciones o cualquier tipo de aplicaciones informáticas, está pensado para que los usuarios de Docker puedan almacenar sus imágenes personalizadas. En este caso, el usuario es “Joyzoursky” y la imagen corresponde a una distribución *Alpine* de Li-

Código 3.3: Código correspondiente al *Dockerfile* de la web.

```
1  FROM joyzoursky/python-chromedriver:3.7-selenium
2
3  RUN apt update
4  RUN apt install default-jre unzip -y
5
6  RUN mkdir /home/webphishing
7  RUN mkdir /home/webphishing/capturas
8  COPY . /home/webphishing
9  WORKDIR /home/webphishing
10
11 RUN apt-get install -y python3-pip
12 RUN pip3 install --upgrade pip
13
14 RUN pip3 install -r requirements.txt
15
16 ENV BROWSERMOB_PATH "/home/webphishing/browsermob-proxy-2.1.4/bin/browsermob-proxy"
17
18 RUN chmod u+x /home/webphishing/browsermob-proxy-2.1.4/bin/browsermob-proxy
19
20 # run the command
21 CMD ["python3","/home/webphishing/app.py"]
```

nux con Python instalado, la librería *Selenium* y la herramienta *Chromedriver* de versión 3.7, recursos indispensables para hacer funcionar la parte correspondiente al análisis de las URLs.

Después, la instrucción RUN, sirve para ejecutar comandos dentro del entorno creado. En este caso ejecutamos comandos como “apt” para la instalación de herramientas, “mkdir” para la creación de directorios o “chmod” para cambiar los permisos de algunos ficheros. Es importante no confundir los comandos de UNIX como, por ejemplo, copiar “cp” y la macro que utiliza Docker COPY. La primera de ellas se utiliza para copiar un fichero por ejemplo dentro del nuevo entorno y la segunda está pensada para copiar un fichero de la máquina host al contenedor. El objetivo con estas cuatro sentencias de la línea 6 a la 9 es crear los directorios requeridos en el contenedor y trasladar en ellos todos los archivos necesarios para la ejecución del software.

A continuación, se instala la versión 3 de Python y todas las dependencias necesarias con las versiones en las que está implementado el software. Después se establece una variable de entorno mediante el comando ENV y finalmente se define el CMD, que es el comando que se va a lanzar cuando se ejecute la aplicación.

3.3. Ejemplos y resultados

En este apartado se muestran con ejemplos, cómo debería utilizarse el software. Como ya se ha explicado, la idea es que existan unos procesos que capturen URLs potencialmente fraudulentas de diversas fuentes y enviarlas al sistema para ser analizadas. Como estos procesos no están implementados, estas URLs maliciosas van a ser seleccionadas manualmente.

Queremos almacenar información de la compañía Dropbox. Para ello, vamos a insertar en nuestra base de datos, varias URLs de páginas fraudulentas que se han encontrado de Dropbox en OpenPhish. Además, se añadirá la página legítima, ya que también nos aporta información sobre posibles recursos que puedan utilizarse en páginas falsas. En el Anexo E.1 vemos algunas de estas páginas fraudulentas. Una de ellas es idéntica a la página legitima. Todas estas imágenes son las que ha capturado el sistema al analizar cada una de las páginas para el almacenamiento de información. En la figura 3.3

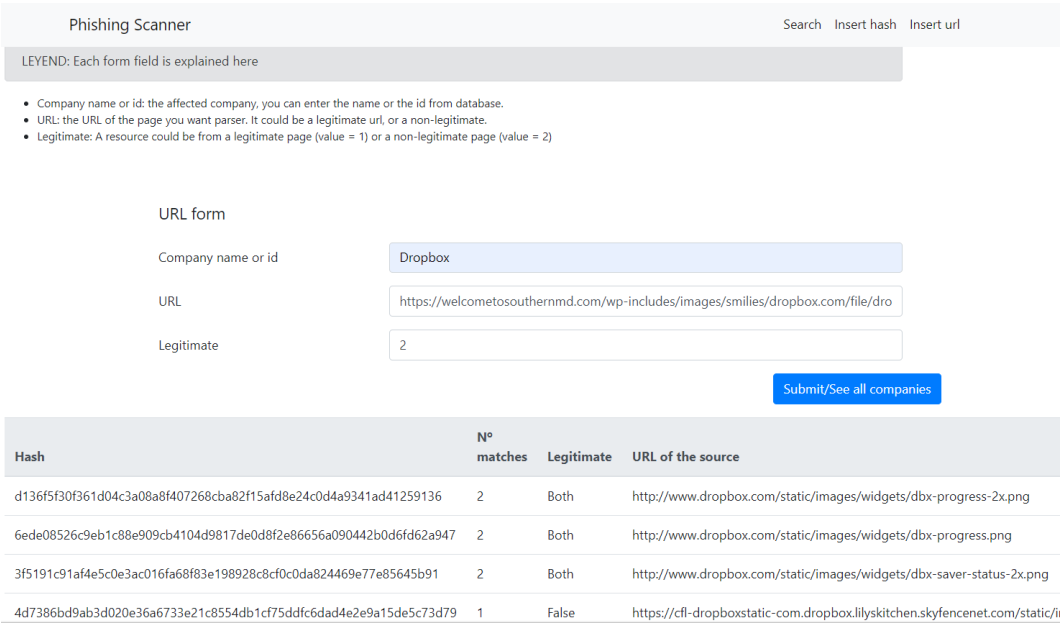


Figura 3.3: Inserción de una URL fraudulenta de la compañía de Dropbox en el formulario correspondiente.

Asimismo, existe un hash de un repositorio de Internet que no sabemos a qué recurso pertenece, pero tenemos constancia de que ha aparecido en otras ocasiones en phishing de Dropbox. Por lo que vamos a cumplimentar el formulario de “insert_hash” para insertar este nuevo hash. Como no sabemos ninguna URL a la que esté asociado el recurso, nombraremos este con una cadena que consideremos identificativa provisional en el formulario.

El sistema está implementado de forma que si en un futuro un análisis de una página web captura el mismo hash, como esta vez tendrá una URL asociada, se sustituirá el nombre asignado provisionalmente por dicha URL. Asimismo, si conociésemos la URL de un recurso web, pero no tenemos en ese momento el hash, puede rellenarse el formulario con la URL del recurso sin el hash, pues el propio

sistema ya se encargará de calcularlo. Así pues, en la figura 3.4 insertamos este hash.

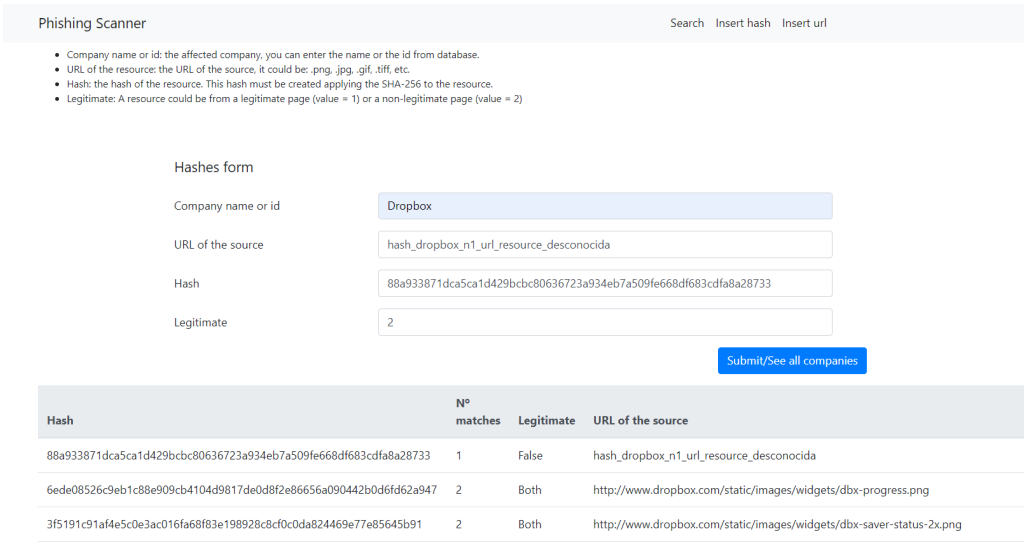


Figura 3.4: Inserción de un hash concreto fraudulento de la compañía de Dropbox en el formulario correspondiente.

Tras haber introducido suficiente información en la base de datos, vamos a hacer una consulta de una página fraudulenta. A lo largo del tiempo aparecen cientos de páginas que suplantan a Dropbox, lo que no quiere decir, que cada una de estas páginas tengan un diseño diferente. De hecho, la mayoría están creadas a partir de las mismas plantillas y kits de phishing, por lo que, almacenando información de unas pocas páginas diferentes, vamos a dar alcance a miles de phishing de Dropbox. En la figura 3.5 observamos un ejemplo de los resultados que aparecen en los análisis.

Los resultados nos indican que diecisiete recursos han sido capturados de esa página web. Han “matcheado” cuatro recursos con Dropbox, otros trece no han sido encontrados en la base de datos. De los cuatro recursos coincidentes, dos han aparecido tanto en páginas oficiales como en ilegítimas y otros dos han “matcheado” con hashes de recursos que han sido capturados de webs fraudulentas.

Puede ocurrir que introduzcamos una página web de Dropbox (en el ejemplo de la figura 3.6 y no consigamos ningún “match” con los hashes de la base de datos por carencia de datos suficientes (desde el punto de vista del almacenamiento).

Los anexos E.3 y E.4 son dos imágenes más del frontal web del software, concretamente en la pestaña de “insert-hash” que no se han mostrado en detalle en este apartado. Asimismo se muestra otro ejemplo de ejecución, esta vez de PayPal. Este ejemplo es interesante puesto que la URL insertada correspondía a un phishing, sin embargo, a los pocos segundos redirigía al a página legítima de PayPal, por lo que la captura ha sido hecha de la página fraudulenta, pero parece que los recursos los ha tomado de la legítima. En los anexos E.5 y E.6 se muestra un anális tanto del phishing como de la la página legítima. El hash “matcheado” resulta ser el mismo.

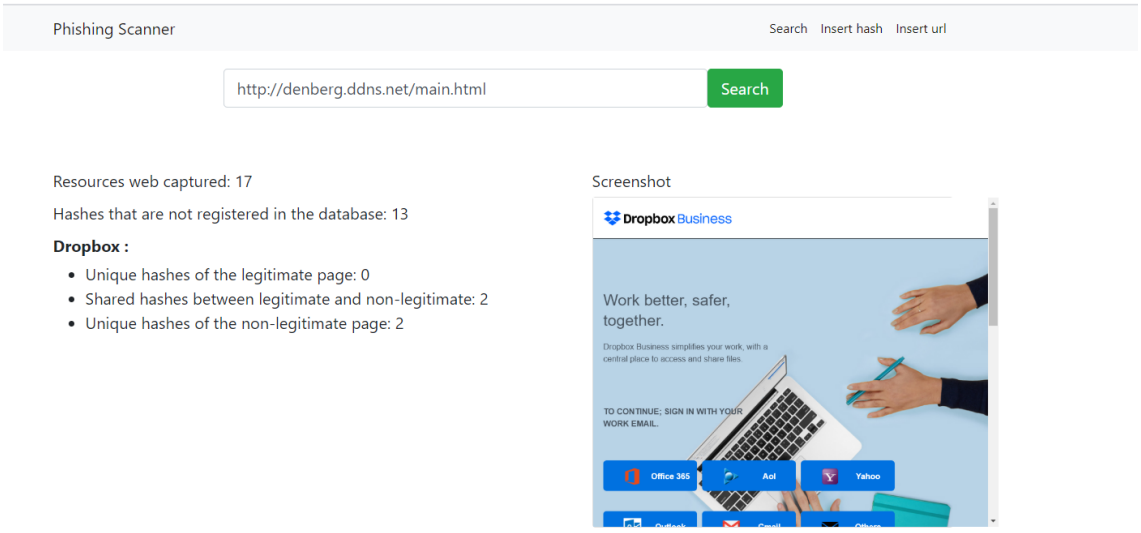


Figura 3.5: Consulta de una URL que suplanta a Dropbox.

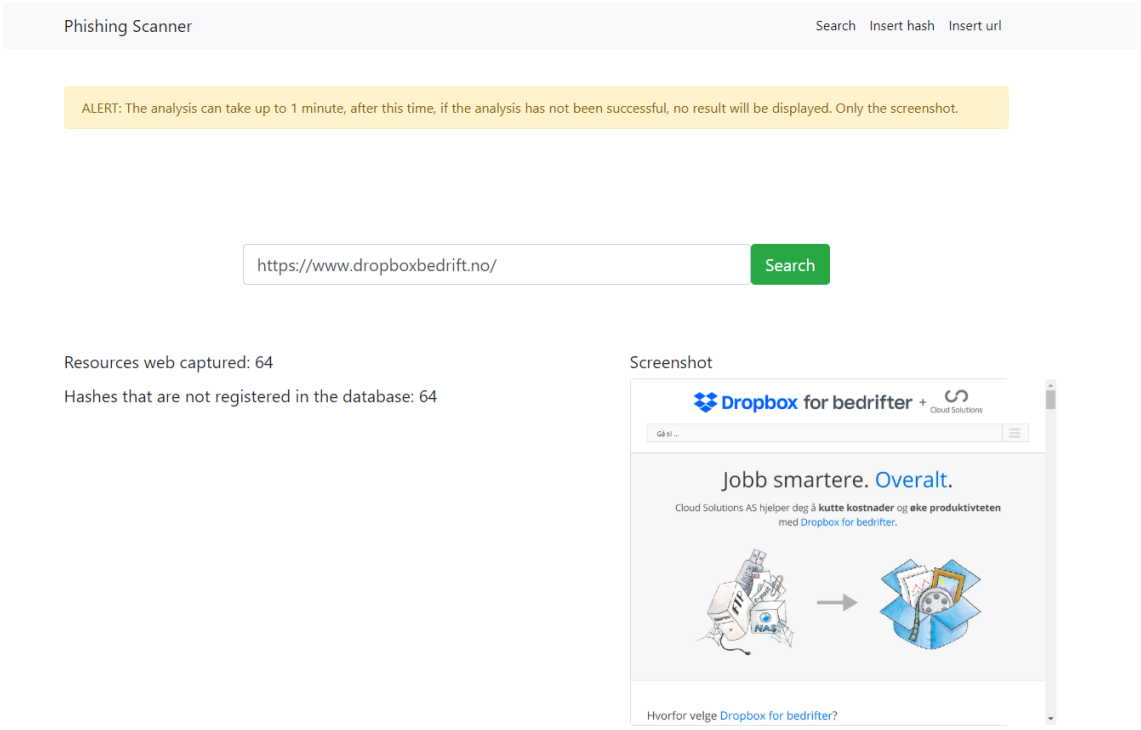


Figura 3.6: Consulta de una URL legítima de Dropbox, pero sin resultados.

En cuanto a la función del hilo “thread_screenshoot” creada con el propósito de reducir el tiempo de resolución de una URL, debido al largo tiempo que dura la captura de los recursos web en proporción a la captura de pantalla, apenas conseguimos una mejora del 10 % del tiempo. En un principio, se pensaba que si la duración de ambos procesos era parecida, podríamos aligerar el tiempo de ejecución en un 50 %; sin embargo, no se ha conseguido una mejora sustancial.

Asimismo se realizaron una pruebas que consistían en consultar varias URLs de forma simultánea. Los resultados fueron negativos debido a que se sobreescriben las variables del código.

El objetivo del Redis y las colas, a parte de reducir la espera de las resoluciones de las consultas, era dedicar a cada consulta un nodo sin que hubiese riesgo de sobreescritura de variables.

Por tanto, ya que no se ha conseguido finalmente montar el sistema distribuido, para solventar este problema, se han colocado unos semáforos en el código de la aplicación “app.py” con los que, al menos, conseguimos que puedan hacerse varias peticiones simultáneas al sistema sin que haya sobreescritura, aunque se vayan resolviendo secuencialmente. Estos semáforos pueden osbervarse en el código de la “app.py”, en el anexo C.1.

4

Conclusiones

Partiendo del trabajo realizado: la comprensión del problema, la búsqueda de la solución, el desarrollo de la investigación y la implementación del software, tenemos claro qué ha funcionado, qué se precisa mejorar y cuáles son las expectativas de evolución del software siguiendo la misma línea de investigación.

En cuanto al phishing, como fraude digital más extendido entre los ciberdelincuentes, no pensamos que en los próximos años se vayan a reducir los casos reportados. Es verdad que a lo largo de la historia del phishing 2.2, ha habido subidas y bajadas de la frecuencia de los ataques de phishing reportados, pero la conclusión que extraemos es que la tendencia va a seguir siendo creciente y, por tanto, la ciberseguridad debe ser rigurosa con este problema y mantenerse constantemente alerta y actualizada conforme evolucionan las técnicas de esta actividad delictiva.

El primer objetivo que se planteó fue crear un software que tuviese la capacidad de recibir URLs para que fueran analizadas con el fin de clasificarlas como fraudulentas o legítimas e identificar la compañía que suplantaban. Así pues, se diseñó un sistema que disponía de una base de datos, un frontal web, una API, y unos módulos de análisis con los que realizar esta tarea.

Del mismo modo, se propuso como objetivo construir una arquitectura base que tuviese la capacidad de dar escalabilidad horizontal a este sistema a partir de múltiples nodos gestionados por colas, utilizando un servidor Redis. Asimismo, la intención era que pudiese ser instalada o migrada en cualquier equipo o máquina. Para ello, utilizamos la tecnología Docker, la cual nos permitía explotar nuestro software en cualquier sistema operativo que tuviese instalado esta herramienta.

De estos dos objetivos se ha conseguido crear una configuración en Docker para poder migrar e implementar el software en cualquier ordenador utilizando un *Dockerfile*. Sin embargo, debido a algunas dificultades surgidas en la etapa final de implementación, y en consecuencia al escaso margen de tiempo que restaba, la parte que correspondía al sistema distribuido para montar una arquitectura más sofisticada (configuración del *docker-compose*): crear un sistema de colas con ayuda del servidor Redis, una base de datos en un contenedor independiente y crear tantos nodos de análisis como fueran necesarios, no se ha conseguido desarrollar. Así pues, aunque no hayamos logrado este último hito, el sistema está modularizado para ser adecuado con facilidad en futuras iteraciones a este formato

distribuido.

Otro objetivo era reducir lo máximo posible el tiempo de ejecución de los análisis. Para ello, hemos paralelizado en el análisis de una página web el proceso de capturar los recursos y el proceso de captura de pantalla, con el fin de disminuir el tiempo de resolución de una URL. Sin embargo, tras ejecutar las pruebas del software, y constatado el largo tiempo que toma la captura de los recursos en comparación a la captura de pantalla no se ha notado una mejoría significativa en la reducción del tiempo.

El alcance que tenía este trabajo no permitía crear un banco de pruebas masivas con las que obtener evidencias estadísticas más completas que nos ofreciesen datos suficientes con los que valorar la eficiencia del software de forma precisa. Sin embargo, en el contexto en el que se han hecho las pruebas, es decir, en un sistema aún primitivo para lo que se busca conseguir en un futuro, la decisión de utilizar hashes como elemento de identificación y clasificación de páginas web la consideramos realmente positiva. Es cierto, que durante el periodo de pruebas muchas de las páginas web consultadas apenas tenían recursos válidos con los que compararlos contra la base de datos. Esto nos hace pensar que sigue siendo necesario complementar la técnica de los hashes con otras explicadas en el apartado 2.1.3.

4.1. Línea de investigación futura

A partir del software desarrollado, se van a nombrar algunas alternativas o soluciones para dar continuidad al proyecto, mejorándolo.

En primer lugar, como la parte de la implementación del sistema distribuido no ha sido finalmente satisfactoria, se podría comenzar por lo siguiente. Ahora mismo, tenemos en un solo contenedor de Docker una base de datos, el cliente web, la API y el módulo de análisis. Para poder ofrecer esa escalabilidad al sistema, se necesita independizar en contenedores diferentes cada uno de los módulos involucrados: el cliente web, los nodos de análisis, la base de datos y el servidor Redis para ayudar con la gestión de las colas.

Ahora mismo, se imprimen una serie de resultados u observaciones hechas por cada URL consultada, sin que resulten inicialmente concluyentes. La intención es que uno pueda deducir a partir de estos resultados si se trata de un phishing o no y a qué empresa afecta. Por tanto, a partir del atributo “matches” que hemos incluido en la tabla “hashes” de la base de datos, se podría crear un *fit* con el que conseguir un resultado más preciso. Es decir, esa deducción que dejamos abierta al usuario, habría que definirla de alguna manera a partir de un algoritmo utilizando el atributo “matches”:

Mencionamos la necesidad de implementar un proceso encargado de capturar URLs de diferentes fuentes de Internet, para que sean enviadas al sistema de forma automatizada. La implementación

de estos procesos no eran un requisito definido dentro del alcance de este trabajo, por tanto, una mejora indispensable sería la creación de dichos procesos para facilitar aportación constante de URLs potencialmente maliciosas y su posterior análisis por el sistema por el sistema.

Otra mejora implica al frontal web, concretamente a la tabla que aparece en los resultados de las consultas donde se imprimen los hashes matcheados junto a otros valores. Ahora mismo es una simple tabla, pero para hacerlo más sofisticado podría crearse una vista dinámica de los datos facilitando criterios de ordenación de la información, a gusto del cliente.

En relación con esto, también se podría incorporar facilidades de gestión de los datos contenidos en la base de datos, como eliminar los datos de una compañía, o borrar un hash específico, etc. Asimismo, se podría añadir una tabla más en la base de datos que almacenara los kits de phishing. Ahora mismo, disponemos de dos tablas: compañías y hashes, pero no tenemos en cuenta los kits de phishing de los que provenían los hashes sino únicamente a la compañía que suplantaban. Por tanto, otra mejora sería la inclusión de esta tabla.

Una tecnología interesante para complementar los análisis de las páginas web, sería el “deep learning”, de procesamiento más costoso, pero de resultado más efectivo. Podría aplicarse un algoritmo de redes convolucionales de reconocimiento de imágenes para identificar el logo, diseños de fondo, distribución de los elementos de la página web, colores, etc. Se podría utilizar hilos para ejecutar en paralelo los distintos análisis aplicados.

Bibliografía

[1] M. Drollet, “The rise of mobile phishing attacks and how to combat them,” 2018. (Visitar).

[2] PhishMe, “Enterprise phishing susceptibility report,” tech. rep., Jan. 2016. (Descargar).

[3] S. Bocetta, “Aviso: Kits de phishing avanzados disponibles en la dark web,” 2019. (Visitar).

[4] C. Cimpanu, “Phishing kit prices skyrocketed in 2019 by 149 %,” 2020. (Visitar).

[5] G. J. W. S. Irani, D. and P. C., “Evolutionary study of phishing,” p. 2, 2008. (Descargar).

[6] R. G. R. J. Chaudhry and S. Chaudhry, “Phishing attacks and defenses,” p. 5, 2016. (Descargar).

[7] C. Ohaya, “Managing phishing threats in an organization,” pp. 1–2, 2006. (Descargar).

[8] M. Seguridad, “El phishing es cada vez más sofisticado y difícil de esquivar.” (Visitar), abril 2017.

[9] H. H. D. Y. Ke Tian, Steve T.K. Jan and G. Wang, “Needle in a haystack: Tracking down elite phishing domains in the wild,” p. 1, 2018. (Descargar).

[10] C. K. Marco Cova and G. Vigna, “There is no free phish: An analysis of “free” and live phishing kits,” pp. 4–5, 2008. (Descargar).

[11] APWG, “Phishing activity trends reports,” tech. rep., 2020. (Visitar).

[12] APWG, “Phishing activity trends reports 1st quarter 2020,” tech. rep., 2020. (Descargar).

[13] M. H. Fernández, “Fraude digital: Prevención, detección, análisis y eliminación.,” 2015. (Descargar).

[14] C. K. Marco Cova and G. Vigna, “What is email envelope and email header.” (Visitar), 2019.

[15] K. O. RG Brody, S Kern, “Anti-phishing: Best practices for institutions and consumers,” pp. 11–12, 2004. (Descargar).

[16] S. Görling, “An overview of the sender policy framework (spf) as an antiphishing mechanism,” 2007. (Descargar).

[17] B. Leiba and J. Fenton, “Domainkeys identified mail (dkim): Using digital signatures for domain verification,” pp. 1–3, 2007. (Descargar).

[18] P. H.-B. T. Hansen, D. Crocker, *DomainKeys Identified Mail (DKIM) Service Overview*, 2009. (Descargar).

[19] Gatefy, “What are rua and ruf in dmarc?,” 2020. (Visitar).

[20] B. Jones, “Gdpr’s impact on dmarc data collection,” 2018. (Visitar).

[21] Boostrap.

[22] Patrick, *Flask web development, one drop at a time*, 2010. (Visitar).

[23] Docker, *What is a Container?* (Visitar).

[24] Docker, *Docker overview*. (Visitar).

[25] Docker, *Best practices for writing Dockerfiles*. (Visitar).

[26] *BrowserMob Proxy. A free utility to help web developers watch and manipulate network traffic from their AJAX applications.* (Visitar).

Apéndices

A

Organización del proyecto software: directorios, módulos y ficheros

Este proyecto, no entra en el perfil de aquellos que requieren gran cantidad de ficheros, con numerosas clases, y archivos secundarios. Se trata de un proyecto Flask, que comprende bastantes herramientas diferentes: bases de datos, logs, cliente web, API, Redis, utilidades, etc.

Cada una de estas herramientas requiere una pequeña implementación, unas pocas líneas de código. Al final, el trabajo consistía en utilizar e integrar distintas herramientas que aportaran un valor diferente, sin profundizar mucho en el desarrollo de cada una.

Todo está organizado como un proyecto Flask estándar. Esto es que está formado por una serie de carpetas que son comunes a todos las aplicaciones Flask, como “templates” y “static” y un archivo primario en Python denominado “app” que recoge las vistas de la aplicación. Este proyecto se puede organizar en archivos primarios y secundarios y en un conjunto de directorios y módulos con otros ficheros.

Comenzamos con los archivos primarios y secundarios del primer nivel.

Dentro de los primarios

- “app.py”: como se ha mencionado, es donde se almacenan las rutas de la aplicación web y la correspondiente funcionalidad de cada una.
- “*Dockerfile*”: se definen el contenido y la configuración del contededor web. Para entender la parte técnica de los Docker, puede consultarse el apartado 3.2.5.

Dentro de los secundarios

- “requirements.txt”: se definen todas las dependencias junto con las versiones de cada una que necesita el contenedor web para que funcione.
- “phishing_parser.log”: es el log que se instancia en el archivo “analize.py” del directorio de la API, y que se usa después tanto en “analysis.py” como en “parser_request”, ambos dentro del módulo de utilidades. Recoge información sobre las peticiones realizadas y sobre los análisis realizados.

- “bmp.log”: es el log perteneciente al “browsermob-proxy”. Se genera automáticamente con la propia herramienta, no ha sido diseñado ni creado por nosotros.

A continuación, se va a dar más detalles del contenido de cada uno de los directorios. Disponemos de los dos directorios propios de la aplicación Flask que son:

Directorios de Flask

- “static”: en ella hay una carpeta “stored” donde se almacenan las capturas de pantalla de las URLs introducidas para los análisis y otra carpeta “temporal”, donde se almacenan las capturas de las URLs consultadas.
- “templates”: se almacena todas las plantillas de la página web. Estas son:
  - “base.html”: de este fichero extienden las demás plantillas, se define lo más básico, esto es el menú de navegación, en el que se puede ir a la página principal donde está la barra de búsqueda o a la página donde está el formulario para introducir un nuevo hash.
  - “index.html”: se trata de la plantilla de la página principal de la web. En ella está la barra de búsqueda donde se introduce la URL. En esta misma página, cuando se recibe los resultados de la petición, se imprime una captura de pantalla de la página web de la URL introducida y los resultados del análisis.
  - “insert_hash.html”: dispone de un formulario para insertar un hash de una compañía previamente creada. También se muestran los resultados en forma de tabla de los hashes que tiene la compañía indicada en el formulario.
  - “insert_urlL.html”: dispone de un formulario para insertar los recursos web de una URL (legítima o ilegítima). Como ocurría en el anterior caso, se muestra una tabla con los hashes de la compañía introducida en el formulario.

Browsermob Proxy

Existe una carpeta dentro del proyecto llamada “browsermob-proxy-2.1.4” la cual fue descargada de Internet [26], y dispone de toda la funcionalidad requerida para hacer las conexiones con los servidores haciendo de proxy donde se alojan las páginas web de las URLs solicitadas con el objetivo de capturar todos los recursos.

El fichero clave de esta carpeta es “browsermob-proxy” situada dentro de la carpeta “bin”. De este fichero necesitamos tanto darle permisos de ejecución como establecer una variable de entorno BROWSERMOB_PROXY para que pueda funcionar el contenedor web.

Módulo de utilidades

El proyecto dispone de un módulo “utils”, donde se recoge buena parte de la funcionalidad del software.

- “analysis.py”: fichero pensado para abarcar todas las funciones relacionadas con el análisis de las páginas web. Contempla únicamente la funcionalidad orientada al análisis de las webs: compara los hashes de los recursos de una web con los hashes almacenados en la base de datos y a partir de unas reglas, estima la probabilidad de que sea phishing.
- “parser.py”: fichero pensado para almacenar todas la funcionalidad correspondiente a “parsear” contenido de la web. Se definen todas las funciones encargadas de iniciar la conexión con el servidor proxy para, la conexión con “chromedriver” utilizando “selenium”, la captura de los recursos de la web y hacer capturar de pantalla.
- “methods.py”: fichero pensado para cubrir todo tipo de funcionalidad secundaria con un fin muy concreto y que pueda ser utilizado por varios módulos diferentes. Los dos métodos de los que dispone son:
    - “render_template”: método que utiliza “app.py” para renderizar las plantillas utilizando Jinja2.
    - “get_hash”: método que utiliza “parser_request.py” para recuperar aplicar la función hash de un recurso web. Se puede elegir qué tipo de función hash aplicar como argumento, aunque por defecto el “SHA-256”.

API

Este directorio guarda todos los ficheros que abarcan la funcionalidad de la API.

- “views.py”: recoge todas las rutas de la API con la funcionalidad correspondiente.
- “analyze.py”: encargado de capturar los recursos de una web y de analizarlos contra la base de datos.

B

Cómo ejecutar la aplicación

Como se ha mencionado en el trabajo, para explotar la aplicación en cualquier equipo solo requerimos de la instalación de la tecnología Docker en el mismo. Después, solo tendremos que ejecutar un par de comandos. Para simular el entorno donde hemos hecho las pruebas, recomendamos probarlo en un sistema operativo linux.

Para instalar Docker, podemos acceder a disitintas páginas web. En el *readme* del proyecto recomendamos “https://docs.docker.com/engine/install/ubuntu/”.

Antes de ejecutar los siguientes comandos es necesario acceder primero a la carpeta o repositorio donde guardamos la aplicación ya que necesitamos estar al mismo nivel de donde se encuentra el Dockerfile. A continuación, introducimos los comandos:

El primero de ellos tiene el objetivo de construir el docker, ejecutar paso a paso las instrucciones del Dockerfile:

Código B.1: Código correspondiente al primer comando para crear el contenedor web.

```
1  docker build -t web .
```

El segundo sirve para ejecutar la aplicación una vez que tenemos el docker web creado y la aplicación instalada dentro.

Código B.2: Código correspondiente al segundo comando para levantar la aplicación.

```
1  docker run --rm --network="host" web
```

El parámetro –network=“host”: indica que la red en la que va a ejecutarse el docker va a ser la del propio host, así podemos acceder desde el host a la aplicación levantada en el interior del docker. Otra opción era exponer los puertos.

C

Códigos de Python

C.1. Fichero app.py

C.2. Fichero analisys.py

Código C.1: Código correspondiente a una de las vistas de la aplicación.

```
@app.route('/')
@app.route('/index', methods=['GET'])
def index():
    dic = None
    url = request.args.get('search_url', None)
    name_img = None
    if url:
        # se baja el semáforo
        sem.acquire()
        # se hace una peticion post a la API a la función encargada de consultar una url.
        data = requests.post("http://localhost:5000" + url_for("api.post_search_url"), json={"domain":
            url})
        dic = json.loads(data.text)
        name_img = dic['image']
        # se sube el semáforo
        sem.release()
    return render_template('index.html', results=dic, img=name_img)

@app.route('/insert-url')
@app.route('/insert-url', methods=['POST'])
def insert_url():
    results = None
    first = ""
    second = ""
    third = None
    if request.method == 'POST':
        # se baja el semáforo y se recuperan los campos del formulario
        sem.acquire()
        company = request.form.get('company')
        url = request.form.get('url')
        legitimate = request.form.get('legitimate')
        # si ha introducido una compañía se inserta en la base de datos la nueva información.
        if company != "":
            requests.post("http://localhost:5000" + url_for("api.post_company_url"),
                          json={"company": company, "url": url, "legitimate": legitimate})
            data = requests.get("http://localhost:5000" + url_for("api.get_company_info"),
                json={"company":company})
            first = "Hash"
            second = "URL_of_the_source"
            third = "N_matches"
        # si no se ha introducido una compañía se muestran todas las compañías almacenadas en la bbdd.
        else:
            data = requests.get("http://localhost:5000" + url_for("api.get_companies_info"))
            first = "ID"
            second = "Company"
        sem.release()

        results = json.loads(data.text)
        if results == {}:
            results = {"":""}
        elif not third:
            for key in results.keys():
                results[key]['hash'] = results[key]['id']
    return render_template('insert_url.html', hashes=results, first=first, second=second, third=third)
```

Código C.2: Código correspondiente a la función que realiza la comparación de hashes.

```
1   db = DBHandler()
2   dic_results = {}
3   dic_results["companies"] = {}
4   dic_results["others"] = 0
5
6   if len(dic.keys()) == 0:
7       dic_results["flag"] = False
8   else:
9       dic_results["flag"] = True
10
11  for hash in dic.keys():
12      query = "select␣name,␣matches,␣legitimate␣from␣hashes,␣companies␣where␣companies.id␣=␣
            companyid␣and␣hash␣=␣?"
13      try:
14          response = db.execute_query(query, (hash,))[0]
15      except:
16          response = None
17      if response == None:
18          dic_results["others"] += 1
19      else:
20          if response["name"] in dic_results["companies"].keys():
21              dic_results["companies"][response["name"]]["num_matches"] += 1
22          else:
23              dic_results["companies"][response["name"]] = {}
24              dic_results["companies"][response["name"]]["num_matches"] = 1
25              dic_results["companies"][response["name"]]["hashes"] = {}
26              dic_results["companies"][response["name"]]["leg_true"] = 0
27              dic_results["companies"][response["name"]]["leg_false"] = 0
28              dic_results["companies"][response["name"]]["leg_both"] = 0
29
30          dic_results["companies"][response["name"]]["hashes"][hash] = {}
31          dic_results["companies"][response["name"]]["hashes"][hash]["URL_resource"] = dic[hash]
32          dic_results["companies"][response["name"]]["hashes"][hash]["matches"] =
                response["matches"]
33
34          if response["legitimate"] == 1:
35              dic_results["companies"][response["name"]]["hashes"][hash]["legitimate"] = "True"
36              dic_results["companies"][response["name"]]["leg_true"] += 1
37          elif response["legitimate"] == 2:
38              dic_results["companies"][response["name"]]["hashes"][hash]["legitimate"] = "False"
39              dic_results["companies"][response["name"]]["leg_false"] += 1
40          else:
41              dic_results["companies"][response["name"]]["hashes"][hash]["legitimate"] = "Both"
42              dic_results["companies"][response["name"]]["leg_both"] += 1
43
44  return dic_results
```

D

Fichero docker-compose

Código D.1: Código no funcional correspondiente al docker-compose. Se muestra únicamente a modo de esquema.

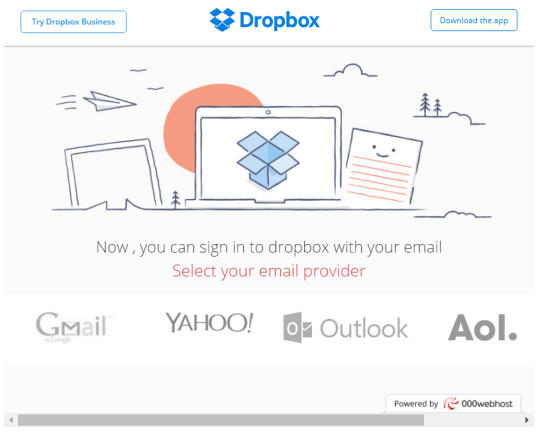
```
1
2  version: '3.3'
3  services:
4
5    web:
6      build: .
7      image: web
8      container_name: web
9      network_mode: host
10     ports:
11       -'5000:5000'
12     volumes:
13       -.:/home/webphishing
14     environment:
15       -FLASK_DEBUG=1
16     depends_on:
17       -redis
18
19   worker:
20     image: web
21     command:
22     volumes:
23       -.:/home/webphishing
24     environment:
25     depends_on:
26       -redis
27     deploy:
28       replicas: 3
29
30    redis:
31      image: redis:4.0.11-alpine
32      ports:
33        -6379:6379
34
35   database:
36     build: utils/database/
37     image: database
38     command:
39     ports:
40
41 volumes:
42   capturas:
```

E

Ejemplos de análisis

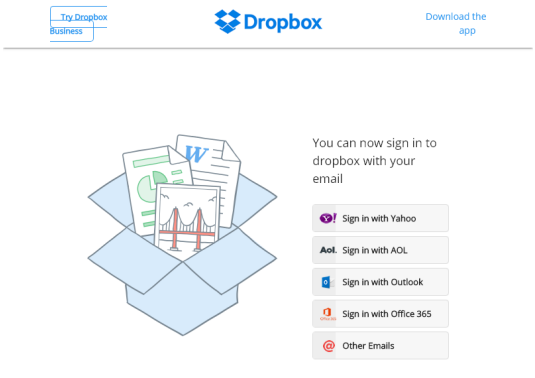
E.1. Páginas fraudulentas de Dropbox

(a)

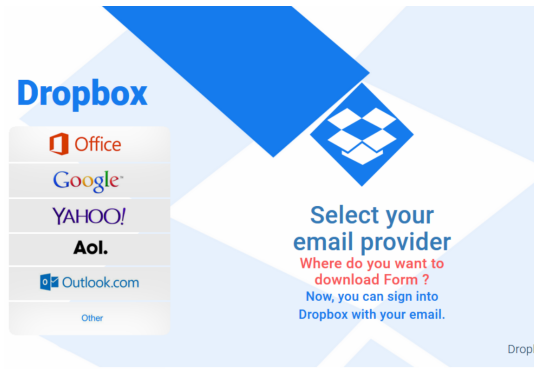


(b)

Figura E.1: Ejemplos de páginas de Dropbox fradulentas parte 1



(a)



(b)

Figura E.2: Ejemplos de páginas de Dropbox fradulentas parte 2

E.2. Otros resultados e imágenes del frontal web

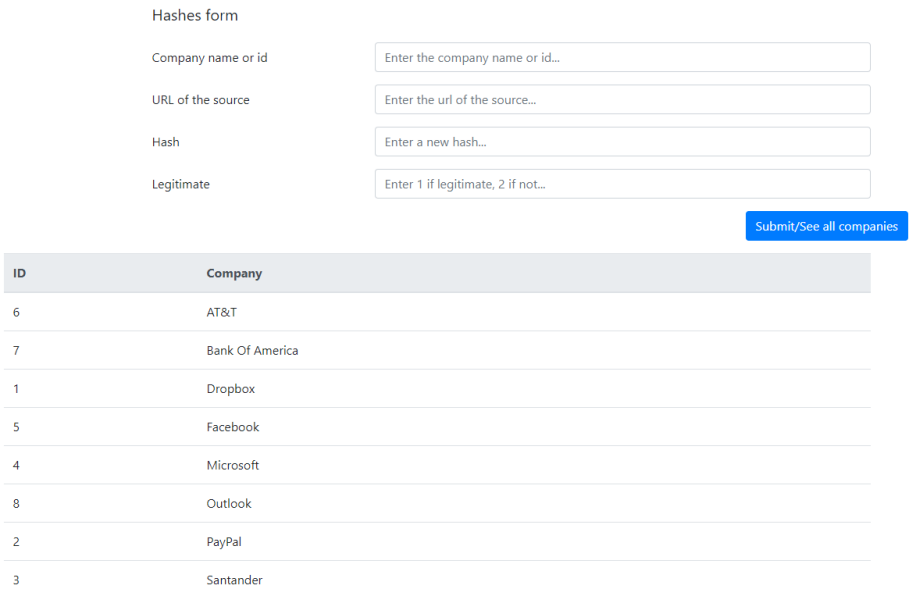


Figura E.3: Resultados del formulario insert-hash al hacer click en “Ver compañías” sin cumplimentar ningún campo.

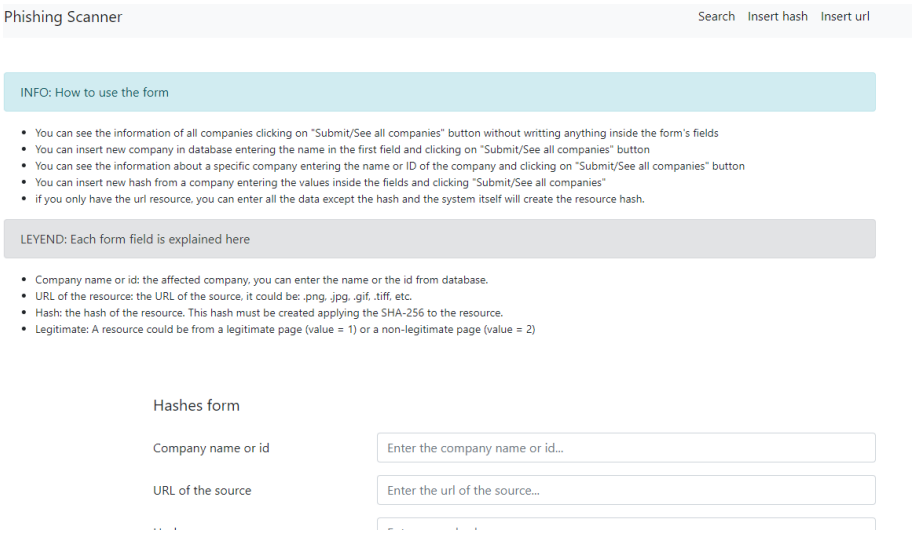


Figura E.4: Información de cómo utilizar el formulario y leyenda de los campos del formulario.

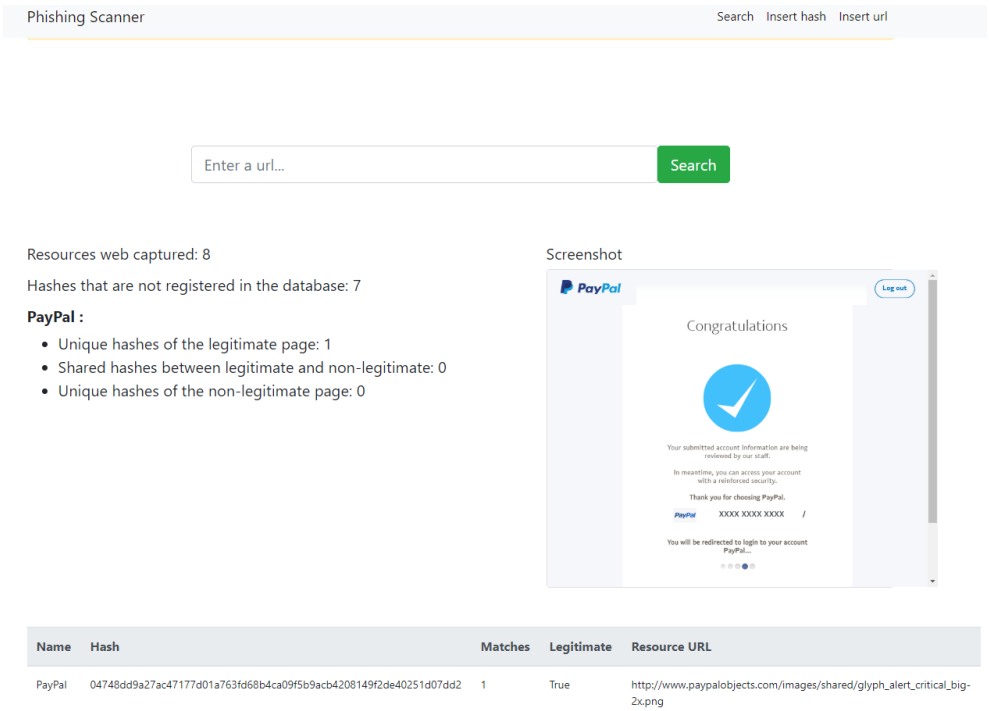


Figura E.5: Análisis de una web fraudulenta que suplanta a PayPal que redirecciona a la página legítima

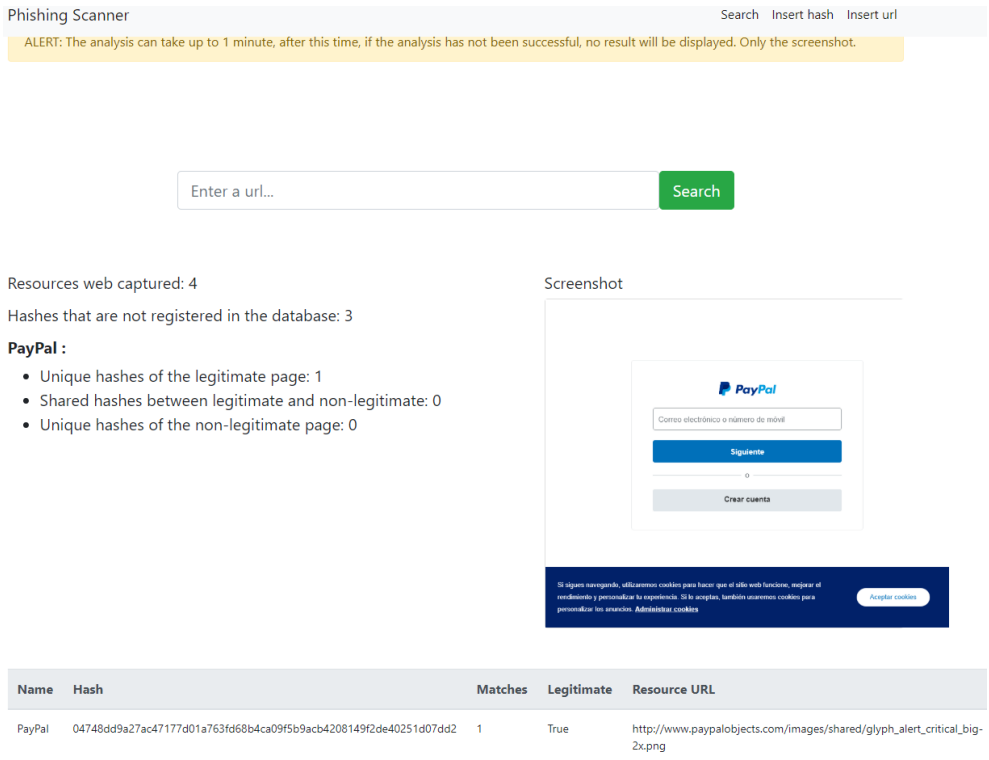


Figura E.6: Análisis directo de la web legítima de PayPal a la que rediccionó la consulta E.5.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID